Contents

Part 0: What are you reading right now

Let us discuss the question: What are you reading right now?

In fact, it may be a very good idea for you not to read any further. This may very well be 157 pages of text that is of no interest to you.

Another answer:

You are reading my entry to the “Summamutikka” competition in the demoparty Instanssi 2023. What that is… find out, and be there with us the next time.

Idea in the particular competition is to “cause confusion” and I can try to do that in the elevator talk part in the live event.

Yet another answer:

This is a preview of one part of a production that I hope I will make publically available in the future.

Author: Yours truly, Paavo, “The Old Dude” of Instanssi.

Part 1: An innocent beginning

My hobby project: Learn Jonesforth, and implement a similar thing.

2019-07-18T21:45 start looking at jonesforth.S

Skipping the links as of now.

I want to implement everything by myself, using NASM, at the pace I’m reading and learning the intro.

I want to make everything super minimal. My driving force is the demoscene intro hobby. I want small.

Remembering the terminology: word, dictionary, direct threaded execution.

Sidetrack: https://forth-standard.org/standard

Read 575 lines and skimmed through std 1-6 and D2.

2019-07-19T00:50 (3.25 hours)

Early history of what transpires here

These first notes were note structured with subsection titles. [Noted 2023-03-03 while contemplating if this could be a "Summamutikka compo" entry at Instanssi 2023 or not…]

2019-07-19T06:30

Initial commit to private git repo: minimal hello world. Continuing with Jonesforth. Read, understood (I guess) and implemented up to line 688 of jonesforth.S Implemented EXIT from standard core. So it begins.

Skimmed through the whole jonesforth.S and jonesforth.F, to fathom the amount of work ahead..

2019-07-19T11:45 (5.25 hours)

2019-07-24T18:05

Implemented and tried out "LIT", i.e., "(LITERAL)" and "+" and "@" and BASE.

Getting an idea about some of this stuff, I think.. Pretty sure I have now implemented 4 simple core words properly, so technically 3% of the core standard…

And then just reading bits and pieces of the standard.. must have been at least an hour.

2019-07-24T23:00 (5 hours; 13.5 total)

2019-07-25T16:00

Implemented "!" all by myself, looking only at the manual definition. Wow. Simple stuff, most of these little words of the standard. But then… towards the couple of messy ones..

Made sure that I've read and understood everything up to line 1135 of jonesforth.S, i.e., 50%. A lot of stuff up to that needs to be implemented later, but it seems quite straightforward just by looking at the standard glossary of required words.

After this session, 55% of jonesforth.s studied and technically 5% of standard core words implemented.

2019-07-25T17:15 (1.25 hours; 14.75 total)

2019-07-25T23:00

Implemented KEY by just reading a non-buffered byte from stdin..

2019-07-25T23:59 (1.0 hours; 15.75 total)

2019-07-26T09:13

Tried to further this project while our bathroom renovation was being finished.. lots of interruptions and not at all efficient usage of time. No coding, just reading jonesforth.S and Forth standard. Well, no code, but I guess I understand more about Forth. Seems to be “liberal and dangerous”, even as standardized. Jonesforth seems to diverge a bit from 2012 core requirements (no “COMPILE,” impossible to make “DOES>”, “”’ available only in compilation mode, . . .).

2019-07-26T17:15 (8.0 hours; 23.75 total)

2019-07-28T10:55

Read through the complete list of definitions in https://forth-standard.org/standard/notation . . . it all starts to make sense.

2019-07-28T13:40 (2.75 hours; 26.5 total)

2019-07-28T15:55

Same, same, . . . just reading

2019-07-28T18:00 (2.0 hours; 28.5 total)

2019-07-28T20:30

WORD, BL, COUNT, TYPE. Understanding more of jonesforth.S second half.

2019-07-28T23:55 (3.5 hours; 32 total)

2019-07-29T08:40

Implementing FIND (preliminary trial). Trying to figure out the xt concept. Glancing at jonesforth.F for that purpose.

2019-07-29T10:30 (2 hours; 34 total)

2019-07-29T20:01

Implementing and debugging WORD, FIND, EXECUTE, DROP..

2019-07-29T22:27 (2.5 hours; 36.5 total)

2019-07-30T07:59

Four very intensive hours. The brain is buzzing and weary. Preliminary (not by far feature-complete) implementation of the interpreter loop. From now on, I can test all new stuff without hard-coded mock-up codes, just via piping text to stdin, which feels like a real milestone.

2019-07-30T11:47 (4 hours; 40.5 total)

2019-08-06T18:01

Getting back up to speed and micro-optimizing (for readability, too, could be called “refactoring”, I guess).

Implemented “:” and “;” but it doesn’t work for some reason..

2019-08-06T21:05 (3 hours; 43.5 total)

2019-08-07T17:30

Debugging, debugging.. turns out the problem was in FIND. Classic: “jnz” instead of “jz”. Well, now I’m back on track.. now I can can have all tests and mock-ups in plain FORTH, I guess..

Implemented IMMEDIATE.. fixed “;” that was buggy.. time to commit the latest to git.

2019-08-07T19:30 (2 hours; 45.5 total)

2019-08-07T20:30

Implemented, debugged, and tried out a first version of number parsing in assembler. Tedious as f.* and dull as sh.* and error-prone as.. idunno.. once in a Forth-project, you’ll need to do this, I guess. Write once, and then forget until it needs to be optimized or refactored (or fixed… probably buggy the first version is!).

Well, it is soooo nice to be able to write “: CR $a EMIT ;” and see that it is FORTH made in FORTH! I’ll immediately make a commit to git, so that all this tedious debugging was not in vain…

Ok, ok.. I ended up implementing and trying out "*" and “/MOD” just for having more fun after all that funless sh.* debugging.

2019-08-08T00:55 (4.5 hours; 50 total)

2019-08-08T21:02

Went through jonesforth.S once again. No news there, anymore.. it looks like I’m at the point of switching to FORTH and examining the jonesforth.F side of things. I’m expecting a lot of changes to the assembler part, too, to accommodate new learnings.

2019-08-08T23:15 (2 hours; 52 total)

2019-08-09T08:00

Changed WORD to skip control chars, when delimiter is BL.

Added some trivial words to the bootstrap FORTH source, just to feel like getting started.

2019-08-09T09:38 (1.5 hours; 53.5 total)

2019-08-09T17:00

Futile, unconcentrated time. Count it in the total.

2019-08-09T18:00 (1 hours; 54.5 total)

2019-08-10T22:00

Band meeting, fermented beverages, and reduced intelligence. Yet, did implement “”’, “,”, “COMPILE,” and most of IF, THEN, ELSE.

2019-08-11T03:15 (5 hours; 59.5 total)

2019-08-12T21:45

Debugged IF, THEN, ELSE following jonesforth. Requires “,”, “SWAP”, and “-”.. turns out Jonesforth is not totally compliant with Forth 2012.. I’d like mine to be.. our paths are likely to separate soon..

2019-08-13T01:55 (4 hours; 63.5 total)

2019-08-13T08:00

Wow, it messes with my head, these nuances of the standard. It looks like my IF, THEN, ELSE might even work, but their implementation must be re-assessed later. I’m using a hacky-looking thing that might go away after understanding and implementing POSTPONE (?). Jonesforth isn’t gonna help, because it doesn’t implement POSTPONE. Might be a later addition to the standard (?), deprecating earlier stuff.

Now… very, very fast, I need to stop this procrastination and go to work. But it had to be debugged until the bug was found!

2019-08-13T11:55 (4 hours; 67.5 total)

2019-08-13T18:20

Recursion implemented. Then many small and trivial things, so that looking at the percentage of standard implemented would feel like progress :). Self-deception. Hard things ahead, and a lot more to understand for the first time.

Then not very focused time. All sorts of sauna, cooking, eating, and watching TV with one of my two eyes. Still, counting hours because I had one half of the thought in here. Implemented BEGIN, UNTIL, AGAIN. Skimmed through rest of Jonesforth.F to see how much uncharted territory remains. Looks like this “course” will be close to the 5 ECTS size that is standard around our faculty. And one course won’t be enough time to create the whole core Forth.

2019-08-13T00:02 (5.5 hours; 73 total)

2019-08-17T16:30

Prepared a README.md just in case I want to make these codes public in github or something. This kind of activity needs a bit of explanation to go along with it…

comment – I wish)" which messes up my stack, because I’m not erroring on non-found word. Lesson learned for the 100th time in life, don’t be lazy.. Ok, I made a lazy half-fix, but still a fix.. “(comment” will

2019-08-17T19:00 (2.5 hours; 75.5 total)

2019-08-18T01:45

Implemented SPACES. Easy peasy. Added a make-target for running the test through the debugger. A little bit easier to launch debugger now. Also, made the debugger just run the stuff until it hits int3.

2019-08-18T02:20 (0.5 hours; 76 total)

2019-08-18T15:30

In airplane… 11 hours of nothing but time without Internet. Well, it seems that sleeping was a priority action at first.

2019-08-18T16:06 (0.5 hours; 76.5 total)

2019-08-18T17:00

Not very efficient. Food and coffee and filling of immigration forms…

2019-08-18T17:30 (0.5 hours; 77 total)

2019-08-18T19:33

More efficient. The symmetric/floored integer division thing learned, and symmetric one implemented… lots of reading of the standard and also AMD64 Manual Vol3 (the Application Programmer's Manual) about integer division and sign-extension.

During the flight, so far, I've implemented at least '/', '2DROP', '2DUP', 'MOD', 'S>D', 'SM/REM', 'FM/MOD'. 'UM/MOD' Maybe some others, too.

Commit current version to git! So many additions have been made after the last commit..

2019-08-18T23:43 (4.0 hours; 81 total)

2019-08-20T17:21

Updated README.md and polished some things here and there. For sending this to a friend as a complete and workable snapshot. I'm not yet sharing this via github, though.. it's too early.

2019-08-20T18:19 (1 hour; 82 total)

2019-08-24T01:45

On a plane again. Eating and watching thunderstorms in the middle.. Then will have to go to sleep. Too tired to learn more, and it should be night time anyway. Marking some 3 hours of this session - approximately correct.

2019-08-24T04:59 (3 hours; 85 total)

2019-08-24T07:45

Debugging a crash.

2019-08-24T08:30 (.5 hours; 85.5 total)

2019-08-24T14:00

Found the bug, fixed.. (wrong padding of xt because I had forgotten that a counted string contains the count byte, so it pads at +1 byte compared to just the string length). Silly, silly.. moving on.

2019-08-24T14:52 (1 hours; 86.5 total)

2019-08-27T06:00

I have spent enough time on this to get this into my dreams. Two jetlagged nights of understanding more and more about Forth. It is not easy to count the time spent on a project that goes on even while sleeping. I decide to count the awake-hours only.

Today, I implemented a lot of the easy stuff that remains. I'm actually getting very close to getting the whole "easy bulk" done. Soon, only nasty and difficult stuff remain. Well, I decide to equate "difficult" with "interesting" like I tell my students to do :).

Yep, and then I implemented more of the easy ones. I think all the easy ones are now done. 70 % of core is implemented. I expect that the remaining 30 % will require a lot of learning time.

Midway assessment: On the average, it has taken me less than an hour to implement each of the 94 first core words. Most of them were simple, and either Jonesforth or the standard had good examples of the trickier ones. I'm now entering a territory where no direct examples exist. Also, I'll need to fix a couple of things that Jonesforth did in a non-standard way.

A little more than 3 ECTS credit points worth of time spent so far. It is interesting to see where I can get before the full 5 ECTS is spent.

2019-08-27T10:44 (4.5 hours; 91 total)

2019-08-28T07:02

Two hours just reading and studying, over and over, the standard, rationale, test cases, and discussions on https://forth-standard.org/ and some StackOverflow and other internet fora. Trying to wrap my head around the POSTPONE and DOES> things. Building up a strategy to continue with implementation:

- CREATE, ALLOT, CONSTANT, VARIABLE could be a nice next step. Meanwhile fixing HERE to be standard and correct previous code.
- Then some bits that could perhaps be easy after those.. ‘C,’, ‘."’ and stuff. In fact, swipe the table of all things that became easy.
- Then re-assess the situation about the difficult bits and how to proceed.

2019-08-28T9:30 (2.5 hours; 93.5 total)

2019-08-28T21:55

Just reading Chapters 0-3 of Starting Forth (online edition). Basics seem to be clear. My own Forth could be used in most of the exercises and examples, but not all. At least I'm lacking DO, LOOP, and ." which are used early on in Starting Forth.

2019-08-28T23:59 (2 hours; 95.5 total)

2019-08-29T12:30

Lunch break. Read Chapter 4 of Starting Forth. Begin Chapter 5.

2019-08-28T13:00 (0.5 hours; 96 total)

2019-08-29T20:30

Read Chapters 5-8 of Starting Forth. Can't actually do the exercises yet, because my own Forth lacks key words like DO, I, LOOP :). Why spend time in trying other people's Forths when building my own… But I think the reading was useful. I can come back to the exercises later. Now I'm ready to start Chapter 9 which is about implementation details.

Also, about 1 hour of reading fora and some example codes about POSTPONE and related stuff. Some people seem to think it is complicated, and some people think it is simple. I let myself anticipate that it is just normal: complicated until the moment of enlightenment after which it becomes close to trivial… I hope I'm on the way to that moment of enlightenment. The thing is.. you never know how far in the future that could be.

I'm considering if I want to have words like "COMPILE-ONLY" as they have in gforth(?). Oh, but did Ertl mention in some post that future versions of gforth will be different? Can't remember… it's all a mess of dozens of forum posts in the hollow backside of the brain.

2019-08-30T01:58 (5.5 hours; 101.5 total)

2019-08-30T11:00

Lunchbreak. Read the beginning of Chapter 9 of Starting Forth. Very useful. Must come back to this as soon as possible.

2019-08-30T11:30 (0.5 hours; 102 total)

2019-08-30T23:00

Bedtime stories: Chapter 9 of Starting Forth, some parts twice, not skipping one comma, all the time referring to Forth 2012 specification, too. Very enlightening.

I think I must come back to Chapter 9 a couple of times… so many important things that one read is likely not enough. Also, the "editor's notes" about "modern Forth systems" seem very relevant.

2019-08-31T02:00 (3 hours; 105 total)

2019-08-31T18:00

Cooking, watching TV, drinking, and all kinds of weekend things. Makes it very inefficient, if you try to further some project somehow at the same time. Yet, I count these inefficient hours because I sort of hung to the project and did think about it for the whole evening. Managed to code a tentative CREATE in assembler, not yet tested at all. Compiles, but no idea if it works…

2019-09-01T00:08 (6 hours; 111 total)

2019-09-01T13:00

Then, after all that reading, it was very fast to implement CREATE, ALLOT, VARIABLE, and some other now-simple words. 99 core words implemented, so at almost 75% of core. Still, some difficult ones remain. Let's see if I can catch up with average 1 word / hour implementation rate…

2019-09-01T13:30 (0.5 hours; 111.5 total)

2019-09-01T14:30

Quickly implemented and tested >BODY.. back to other weekend stuff.. grocery store and food and stuff..

2019-09-01T14:35 (0.0 hours; 111.5 total)

2019-09-01T16:30

Some more words.. up to 102 in fact.

2019-09-01T17:15 (1 hours; 112.5 total)

2019-09-01T17:45

Trying to fix HERE to comply with the standard.. And fixed it.. I hope. Still lacking proper tests.

2019-09-01T18:10 (.5 hours; 113 total)

2019-09-01T19:05

Hmm… watching TV again while coding.. this is not how to be creative!

2019-09-01T20:05 (1 hour; 114 total)

2019-09-02T22:05

Some more. There is catching up to do if I want to get back to the 1 hour/word average. Trying, trying.. 117 hours and I'm at 110 words after <# HOLD SIGN #> ..

2019-09-03T01:10 (3 hours; 117 total)

2019-09-03T21:20

Hmm… how did we divide back in school? Monospace font needed to be legible::

For example, divide “doublecell” decimal 11*100 + 32 by 3. q = quotient, r = remainder of H = high part, L = low part

```
qH qL
 3 77
```

`-------`

3 | 11 32 -> quotient 377, remainder 1 2 32 rH 1 rL

Yep, this is how we divided back in school… this could work. Check the printing algorithm using # and #S now that I might have double divide OK?

377 -> 125, 2 -> 41 , 2 -> 13, 2 -> 4, 1 -> 1, 1 -> 0, 1

I would do this on paper, but right now I’m separated of such tools.

Finally implemented “#”. Phew… that was a hard nut to crack. I also discovered that the low and high part of my double values were in the wrong order in memory. I always, always, do this.. intuitively pick the incorrect of two opposite options.

Then #S should be trivial, because I have REPEAT and UNTIL Well, it almost was.. but it was easy to get lost with the stack again.

I think I have the <# # SIGN HOLD #S #> package. Now it would be really easy to re-do the number printing using these. I think they would be nicer. But enough time spent on number printing.

Next up: strings. Not tonight, though. I’ll leave it at implementing ENVIRONMENT? that always returns FALSE.. is this permitted, I don’t know.. Anyway, I’ll do it properly later. Now I just want to see some numerical progress again: 113 / 133 core words (85%) implemented. Only 20 more to go. But those are the messiest ones, at least in my own head.

2019-09-04T00:42 (3 hours; 120 total)

2019-09-06T12:30

I’ve been reading Chapter 11 of Starting Forth on and off during lunchbreaks, walking to-and-from work, and other 5-10 minute possibilities. Desperate and unfocused, but I think there has been some development of thought amidst the chaos. I’ll count some +2 hours to the total as an approximation.

Today is a day off work, and a three hour bus trip to parents-in-law. I hope I can be super focused in the bus, and apply the learnings from Thinking Forth. Especially, I’ve learned that POSTPONE and DOES> are quite fundamental in current Forth. Now there is no way around it: I need to build tentative implementations of these two words, and try to re-implement some other words on top of them. Quite simple should be at least ['], VARIABLE, CONSTANT, LITERAL. There may be more to POSTPONE and DOES> than I realize yet, but I really think I can’t grasp them without tinkering and trying to build them.

So, I have a goal, and I have a couple of hours of time without distractions. High hopes.. let's see how it turns out today.

I have no more than 15 hours left in the allotted 5 ECTS credit points I've been planning for. It will be very interesting to see what I can accomplish in the remaining few hours. Of couse, the project continues after that, but it will be one milestone and an interesting point of reflection.

2019-09-06T13:00 (0.5 hours; 120.5 total)

2019-09-06T15:30

. . .

OK.. I have a kludge of a POSTPONE that might work, but possibly not quite.. Anyway, when I get POSTPONE working properly at some point, I believe I already have ['] and [CHAR] copy-pasted from reference implementation embedded in the standard document.

2019-09-06T18:00 (2.5 hours +2; 125 total)

2019-09-06T19:21

2019-09-06T21:00 (2 hours; 127 total)

2019-09-06T22:45

Debugging DOES> .. to find out that it can't be implemented in the way I thought. Better to sleep and try again in the morning.

2019-09-07T01:30 (2.5 hours; 129.5 total)

2019-09-07T11:00

OK. I have a DOES> that plays along with the examples from Starting Forth, Chapter 11. Very, very, cool. But questions remain.. it was as hard as I thought it would be.

Well, next up: The DO loop stuff which I've been avoiding so far.

2019-09-07T11:28 (0.5 hours; 130 total)

2019-09-07T13:35

DO is actually given in Starting Forth, Chapter 11. Very simple.

2019-09-07T14:00 (0.5 hours; 130.5 total)

2019-09-08T14:30

LOOP needed a lot of debugging.. once again because of the impossibility of me getting a constant number right when there are a couple of options. (Basically, jumping from 0 to 3 is +3 and not +2, but even while staring at an equation like 0+2=3 I still think it is correct, until I see the jump go wrong in the debugger... funny thing this trick of the mind... but I can do stuff because there are these wonderful tools like gdb, and pen and paper for sketching and simulating such jumps.)

Ok.. after the return bus trip back from the in-laws, I have DO, LOOP, I, J, UNLOOP. Some re-arrangements are required for +LOOP and LEAVE.

I only have one hour until the 5 ECTS credits is full, so I will now pause the actual work on the project and make some kind of intermediate conclusion. Reflection and wording of thoughts is important, I tell my students. Let me try, as an example.

2019-09-08T17:00 (2.5 hours; 134 total)

2019-09-16T16:40

(————————————————————————)

OK. It is time to make a debrief of my “simulated 5 ECTS self-study”.

Let’s see. I set out to “Learn Jonesforth, and implement a similar thing”. How it went down in 135 hours:

- I think I understand all of Jonesforth. Every bit, every instruction, of the assembler part, and any word of the Forth part that I have yet set my eyes on. I’m not really interested anymore in the parts not in standard Core. So, for any interesting part, it is now learned. Done and dealt with. Grade 5/5 for this goal!
- I didn’t re-implement Jonesforth. I ended up dreaming bigger and attempting to implement a standard Forth 2012 core system. Well, my implementation is fundamentally a “similar thing” to Jonesforth because it is a “classic, indirect threaded execution implementation”.
- 90% of Forth 2012 core words are implemented after 135 hours, and I have a pretty good idea of how to implement the rest. There is a lot of work ahead, but regarding learning, I think I came a long enough way.

  It seems that I came up with this new goal at the exact middle point (73 hours) of the project, noting at the same time that “one course won’t be enough time to create the whole core Forth”. Well, not without prior experience anyway. And a “first course” is always a “first course”.

  If this was a student project, I would accept the current thing as final coursework, even in this unfinished form, should the student also give a proper overview of the remaining issues and a general idea of how the issues could be dealt with. Resolving those would be quite mechanical work and unnecessary, learning-wise. “On to the next project!” I would say to the student…

  Of course I will continue with this one, because I have the bigger crazy goal of making a demoscene entry in Forth. And I really want to be close to what the standard specifies. Meeting these goals involves a lot of “just work" and "dull but necessary refactoring" with diminishing returns hour by hour. Given that nobody pays for such things, I guess it is something for hard core hobbyists, crazy people… let's say it's just for fun?

What would I do differently, if I started this project afresh:

- I would diverge from Jonesforth at an earlier time, thus re-defining the original goal setting. The differences between Jonesforth and the current living standard became apparent at some 30 hours into the project and crystal clear at 60 hours. But the decision to continue with Jonesforth was just a big ALL-THE-SIGNS DUP READ? SWAP UNDERSTAND? AND IF EXIT … I still think that Jonesforth was a very, very good way to really dive into a (traditional) Forth implementation. Perhaps somebody (ehm.. who's got the time and interest?) could make a 2020's re-write of it. Would it be too difficult to make a reader dive into a more modern take of Forth?
- I would start reading Starting Forth earlier. I think it could have made the whole project go smoother and faster.

For me, as a teacher, the most important thing was to re-learn a lot of fundamental things about studying:

- Studying is really hard. It comes as a surprise, every time I get to try it myself :). It is important for the teacher to remember what the student goes through on every single course. And there are so many courses in a degree… one right after the other and many at the same time. Wow, those people are crazy to go through our fully packed curriculum!
- Studying in the short and sparse moments among work and hobbies is even harder. The hardest thing is finding the suitable *kind* of time. Procrastination is a real enemy.
- It is hard to keep focused on a task. Unfocused time is very futile. *Planning for proper learning situations* without distractions is key, I think! (I seem to suck at it. Never able to say "Quiet! I'm studying for two hours now!")
- *Keeping track* of time and the *quality* of the time can be really enlightening. Occasionally, everybody should benchmark the way that they actually use their time.

With these finding in mind, I think I will have to re-evaluate my expectations of what the average student can achieve in a given period of time. I think I may have to cut stock on my course contents. We "only" expect the student to study 8 hours a day, 5 days a week. Well, it is reasonable to expect them to try. But I think it would require a superhuman to keep focused in demanding tasks 8 hours a day.

Where do I want to go from here:

- Finalize and test the DO LOOP -stuff

- Then implement the remaining few words of Forth 2012. This will likely require modifications to the input routines, because they currently model the KEY input of Jonesforth.
- (anticipate some 30 hours of work for the above)
- Revise the whole thing according to current knowledge. (perhaps 10 hours?)
- Then, the interesting part: Hook up SDL2 and OpenGL and make an actual proof-of-concept of demo. Window with triangle and beeping would suffice at this point.
- Meanwhile, start reading Thinking Forth.
- Then, it is very, very important to take a complete break from this project and spend my time on other interesting ones!
- Return to this in the beginning of 2020, and contribute a smashing live-coding entry to Instanssi 2020 "Summamutikka" competition (I hope they have it).
- The crazy dream is then to size-minimize this and make a 4k intro to Assembly Summer 2020 compo with the techniques used field containing "Standard Forth 2012 running on my own interpreter".

So, this concludes my notes on the first 135 hours of learning the "Introduction to Forth". Let's eventually call it notes1.txt and continue in notes2.txt.

2019-09-16T18:07 (1.5 hours; 135.5 total)

Part 2: "Second course in Forth: Implementation of advanced standard words"

(Learning diary of a hobby project.)

Hmm.. why not write these notes in Markdown.

Template

Diiba daaba..

2019-09-16T18:15 - 2019-09-16T18:15 (0 hours; 0 total)

Start

Created this file and added it to the repo.

2019-09-16T18:15 - 2019-09-16T18:15 (0 hours; 0 total)

Kick-off

Wow.. mind-bogglement over LEAVE. First I thought it was going to be simple. Then I thought it was complicated and resultingly did it wrong. Now I again think it is simple, but I'll have to sleep over this and fix it later. I hope I'm right this time. Too many hours spent on one single word. Well, I've discovered a load of important stuff that could be made simpler in the forth-coming (hehe) revision stage. The BRANCH stuff is one, and there are plenty of others.

2019-09-16T20:07 - 2019-09-17T01:15 (5 hours; 5 total)

Proper plan

Mostly copy-paste from the previous notes:

- Finalize and test the DO LOOP -stuff
- Then implement the remaining few words of Forth 2012. This will likely require modifications to the input routines, because they currently model the KEY input of Jonesforth.

  The first implementation will be a kludge.. Goal: live with that.
- (anticipate some 30 hours of work for the above)
- Revise the whole thing according to current knowledge. (perhaps 10 hours?)
- Then, the interesting part: Hook up SDL2 and OpenGL and make an actual proof-of-concept of demo. Window with triangle and beeping would suffice at this point.
- Meanwhile, start reading Thinking Forth.
- Then, it is VERY, VERY IMPORTANT to take a complete break from this project and spend my time on other interesting ones!

[the break would have happened at this point of the planned roadmap; turns out it happened earlier..]

[and it turned out that the break was very long and didn't have any time for other interesting projects; handling the extra work from COVID-19 situation, no time for hobbies; so, the whole plan was practically postponed by one year]

- Return to this in the beginning of 2020, and contribute a smashing live-coding entry to Instanssi 2020 "Summamutikka" competition (I hope they have it).
- The crazy dream is then to size-minimize this and make a 4k intro to Assembly Summer 2020 compo with the "techniques used" info field containing "Standard Forth 2012 running on my own interpreter".

And... yep, I did get LEAVE working. It would be [was?] even simpler than I thought before getting some sleep.

2019-09-17T16:15 - 2019-09-17T17:15 (1 hour; 6 total)

Oh.. trying to make +LOOP but it is not as simple as I thought.. nothing usually is, but still.

2019-09-17T21:30 - 2019-09-17T23:31 (2 hours; 7 total)

Finally, +LOOP works like expected. I did it in assembly to keep it simple.

2019-09-26T21:05 - 2019-09-26T23:48 (2.5 hours; 9.5 total)

It is time to make the input buffer properly. Read standard, and implement >IN and the proper parse stuff.

Ok, but ACCEPT works with KEY and char buffer memory instead of the input buffer, so I'll quickly implement ACCEPT first..

Nah.. going nowhere in half an hour after one week off-project. Longer periods of concentration needed. Where could I find some?

2019-10-05T12:35 - 2019-10-05T13:05 (0.5 hours; 10 total)

Longer time, but not concentrated. Very inefficient. A version of ACCEPT but it crashes.. can't concentrate on debug. Must sleep.

2019-10-06T20:05 - 2019-10-06T22:35 (2.5 hours; 12.5 total)

Yep, after sleeping, with a sharp brain.. no problem there.. I had some thinkos in stack use (wrote a wrong stack comment, and after that all is lost until debugging reveals the mistake.)

Finally, after too many hours of inefficiency, ACCEPT is done.

Next up: Input buffer and parse area according to the standard specification. so.. first off to study the first parts of the standard again.

2019-10-07T09:45 - 2019-10-07T10:15 (0.5 hours; 13 total)

Nah.. tried to sit down with code, but failed completely by doing some other business.

2019-10-13T19:55 - 2019-10-13T20:31 (0.5 hours; 13.5 total)

The break happened.

Then, it turns out, there were other things to do in life, at work and at home. I am now returning back to this after, let's see, 5 months and more than a week. That is practically half a year.

Since then I've started tracking all my hours with a software tool, and I am amazed and happy to be able to track some time tagged as "hobby programming".

Regarding this project - I remember nothing about this after the 5 months break, so I'm starting with reading my own code and documentation and hoping that I have done the documentation well enough last year.

From my original plans, the pre-show at Instanssi 2020 had to be omitted, because it happened during the 5 month silence. Instead, we once again had some workshopping fun with my lmad1 library and Javascript at Instanssi 2020. The Assembly Summer 2020 goal remains as of now, and I turn my hopes towards making a "making-of" presentation in Instanssi 2021.

While at it, I think it is time to put this thing in GitHub as a private repository. There's more than 150 hours of effort to be wasted if my hard drive crashes, so I want this to be safe in the clouds. I'll make the repository public after the first compo show… planning towards Assembly Summer 2020 as of today.

There, done… phew, a lot safer feeling now.

2020-03-22T15:05 - 2020-03-22T15:39 (0.5 hours; 14 total)

Coming back from the break

Wow, these notes are good. After reading this file up to here, I know exactly where I was going with this project half a year ago. Here's the quote from above:

Next up: Input buffer and parse area according to the standard specification. so.. first off to study the first parts of the standard again.

So… I'll come back to exactly this now. Can't really call this "remembering" because the memories came back from these notes projected as text. But it works for the same purpose. I do remember telling my students that it is important to make notes. Now I also remember why I keep telling them that. Great. So some things were in fact remembered here.

Now I go jogging, processing these things in my mind. So trying to multitask in the only way I can, as of yet. I hope to come back to this every couple of days now until I get something to show in a demoscene compo.

2020-03-22T16:37 - 2020-03-22T17:38 (1.0 hours; 15 total)

The COVID-19 happened

I did not come back later that evening nor any couple of days since that particular Sunday in March 2020… Instead, I worked for 130% of my allocated working time. One reason was COVID-19 and the sudden transition to distance learning.

Now it is July and finally summer holidays: jogging and maybe coming back to this project actually every couple of days. And the Assembly Summer compos were postponed due to COVID-19, too, so there's no rush with this project. Maybe still some slight chances of meeting the Assembly/TRSAC 2020 compo deadlines.

I went through the standard specifications of the unimplemented words again. Indeed, the next thing seems to be creation of proper input buffer according to the input specifications. Related words “>IN”, “SOURCE”, . . . , and probably revising my earlier quick-and-dirty implementations of input parsing.

2020-07-15T10:00 - 2020-07-15T11:00 (1.0 hours; 16 total)

It always takes time to pick up the pieces

Ok.. once again I read the earlier code, and the comments I fortunately had put in there. The comments remind me of the roadmap.. (“first implement a Jones-like WORD using KEY, but consistent with the std. Later, after I’ve learned what Jonesforth has to teach me, I’ll be implementing PARSE and PARSE-NAME, and switch to using those.. Molding and re-molding it will be..”).

How nice that I had written these comments. No guessing necessary at this point of time.

But a lot has to be re-learned from the specs. A lot of time will be spent in getting oriented in the current situation of the unfinished project. Slowly getting there..

2020-07-15T15:00 - 2020-07-15T18:00 (3.0 hours; 19 total)

Ready again

One more hour reading the code and the standard. Push to github, to feel like being back in the business. I think I’m ready to start implementing. A little break to refresh the brain - not too long (like months). Coffee.

2020-07-16T07:45 - 2020-07-16T08:39 (1.0 hours; 20 total)

To feel ready is not to be ready

Then there was quite inefficient time. Thoughts were somehow wandering in places, not able to focus on the task. Tried to get an idea of what should be done first, but it was difficult. Did more reading of the old code and the standard. There was some procrastination with the proposals and discussions, too. This is the old enemy, “I will read everything that the Internet has about this before actually picking up the keyboard”.

It seems that I have to count quite a few more hours into the “start doing it” phase. It doesn’t happen in an instant. An honest 10 hours of unfocused warm-up time needs to be added.

I did implement (but not yet test) >IN, REFILL, SOURCE-ID, and some parts of what will become my “input source specification”. Based on the readings, I became certain that I need to tear down all uses of WORD. It turned out to be a long and hard trip to begin by implementing WORD that uses KEY. . . Only dozens of hours into the project did it become apparent that WORD is becoming obsolete. It would definitely be nice to have a 2020s revision of a tutorial like Jonesforth… one that would start by implementing buffered parsing. I'd love to do such a tutorial, but not on my spare time. Too many more interesting projects ahead.

Next thing to do: QUIT should use the new REFILL and PARSE and PARSE-NAME. Use of WORD should be abandoned.

2020-07-16T09:05 - 2020-07-16T19:39 (10.0 hours; 30 total)

Finally warming up

Much better today. Small increments and debugging. I now have an input buffer, and a good idea of how to continue with PARSE and PARSE-NAME. There will be many hours of work because the stuff is actually spread in many places. Turns out that this was not the simplest point of the project to pick up after more than 6 months. Well, I'll just count all these picking-up-periods to the total hours and see when, where, and how this project lands, eventually. Now I'll commit and push a version with all my earlier test cases working.. with initial implementations of REFILL, >IN, and SOURCE-ID. And the first baby steps taken towards standard-compliant parsing and a more elegant interpretation loop.

2020-07-17T12:01 - 2020-07-17T16:04 (4.0 hours; 34 total)

Celebration after a day of debugging

It is now better after all that warming up and picking the pieces.

Last night I implemented PARSE and then played with it for some time. Projects would progress much faster if I could resist this urge to play with every new piece of a tool I get done. But then again, these projects are for fun, and I think that some playing must happen or I will lose interest. To be honest, all that seemingly unproductive playing must be counted in the hours. So, 2 hours from last night of playing and enjoying, after first implementing PARSE of course…

Today, I sat down for a longer time and implemented ." - It required a lot of recapitulation of previous codes and comments and re-reading many parts of the standard. And a lot of debugging using gdb. And the realisation that I still have a very long way to getting things "right" and even understanding what the right way should be.

Needless to say, after all that debugging and painful observations of tasks remaining in the future, I did spend some time playing around with the newly built capacity of actually building "Hello world" -like applications. And writing these project notes, which, for me, is part of the fun.

I'm currently tracking all my time in life using the nice Estonian tool toggl.com so I know how many *minutes* I've been sitting with my laptop with this project today. For these notes, though, it is honest and detailed enough to make an approximation of 6 hours to the total project time that I spent in implementing PARSE and ." and playing and having fun with them.

It is to be noted that these 6 hours were by no means consecutive nor without interruptions from the "real world". It is Saturday, and I tried to have some holiday life also outside of just coding. Nice bicycling around the city and going to second-hand shops with wife, some food and beverages, and so on.. 6 hours without interruptions might have gotten me further than 6 hours in 30 minute fragments, but real life happens to come with interruptions. An important note to self:

I should try to remember this pervasive fact of life also when I try to guess how much time students will need for the assigments I give them, and how they are to be rewarded with ECTS credits. According to specification, 1 ECTS == 27 hours … 27 hours of what? Focused time that really allows learning and creating? What if that comes with another 27 hours of unavoidable interruptions from the environment? Should I think that 1 ECTS == 27 hours of time that the student deliberately *tries* to use in the tasks, but unavoidably fails for most of those hours. Then, how do I manage the difficulty of the tasks… Interesting things to think about.. and dream about.. Now I've spent almost an hour on writing these notes, which pretty much proves the points and leads to the realisation that I should stop right here, commit, and continue tomorrow with a new day and a brain that has slept.. But really think about these requirements to my students later on…

Project status: Implemented 128 core words out of 133

2020-07-18T00:01 - 2020-07-18T23:59 (6.0 hours; 40 total)

Sometimes slow, sometimes fast

Quickly copy-paste-modified the ." to get S" and ABORT". Well, not sure if they are really doing what they should be doing. It doesn't really matter because I know there is a lot to be learned and re-thought after the next big goal. That next big goal is to get a first almost-working version of the whole core word set. Depending on real-life interruptions, I think it could be a matter of just a few *calendar days* now. We'll see.

At the moment, I'm only short of 3 implementations: >NUMBER EVALUATE and SOURCE. It is a small number of features, but I know I'll be looking at a great deal of re-factoring and perhaps even re-designing big parts of the earlier implementations. None of this input source separation and buffered parsing/evaluation was in the scope when the first parts were written. So a lot has to be modified. I hope I can do it one small step at a time, as before.

Project status: Implemented 130 core words out of 133

2020-07-19T00:01 - 2020-07-19T02:07 (2.0 hours; 42 total)

Flowing much better now. >NUMBER implemented, and I did make it in Forth - with juuust a bit of debugging and re-discovering which way ROT works and that IF needs to be written in its place… Oh, what fun. And yes, once again I had to have some fun with experiments, too.. therefore as much as 2 hours working and playing with >NUMBER.

Project status: Implemented 131 core words out of 133

2020-07-19T23:50 - 2020-07-20T01:38 (2.0 hours; 44 total)

So, only two words missing: SOURCE and EVALUATE. These go together, and probably need changes in older codes.

So far there has only been the terminal input from stdin since the first steps of this project. Now it must change to facilitate EVALUATE from string, and I need to implement an "input source specification" that "enables the nesting of parsing operations on the same or different input sources" (quoting the std)

Two more morning hours spent in planning how to go about this change that is going to be a little bit bigger..

2020-07-20T01:38 - 2020-07-20T03:23 (2.0 hours; 46 total)

Day of debugging, but then.. I'm at a milestone!!

Implemented SOURCE as a shallow front for input source specification.

Changed a lot here and there to be able to implement EVALUATE.

And there. It is done!!

2020-07-20T12:26 - 2020-07-20T15:30 (3.0 hours; 49 total) 2020-07-20T16:02 - 2020-07-20T17:07 (1.0 hour; 50 total)

Commit and push new changes as soon as possible. Then make notes of the project status. The thing to celebrate is:

Implemented 133 core words out of 133 !

Self-assessment and review of project roadmap.

Intermediate self-assessment follows.

In numbers: It seems that I was able to make my first (kludgy and likely buggy, but very nearly complete) Forth core system in 185 hours. On the average, it took 1 hour and 24 minutes to produce each of the 133 core words. That average number is quite meaningless, of course, because of the variation. Some simple words took a few seconds whereas the more difficult ones took several hours. Also, the total hours of this project included also learning many things for the first time, reading a textbook, and writing these notes.

The next one would be faster. Would be done in a month of working hours :). Not that I want to start speed-running Forths.

So far, so good. I won't be giving myself grade 5/5 of this part because I have failed in scheduling hours in life. I'm sure I could have spent fewer over-hours at work, watched fewer shows on TV, read fewer news from the Internet, and generally picked up this project more often and with more focused attitude. Self-assessment, based on these key elements, is 3/5. Won't be less than Good because I'm all set up for the next parts. But I have been far from Excellent in self-leadership required by self-directed self-study.

Next up: Get the official test cases running. I'm quite sure that there will be failing tests and a lot of need for debugging. After that, the integration of SDL2 and OpenGL as per original roadmap.

Half an hour for this self-assessment.

2020-07-20T19:40 - 2020-07-20T20:08 (0.5 hour; 50.5 total)

Brief look at http://www.forth200x.org/tests/ttester.fs … That test harness needs further words at least from the Optional Programming-Tools word set. Makes perfect sense, of course. I'll have to decide if I want to use time to implement those programming-tools (fun and useful if I was planning on staying with my system for a longer time). Ok, but it seems to be mostly the floating point stuff.. and I suppose just [IF] [THEN] [ELSE] .. doable, but it would take zero time to just delete the floating point part at first.

Ok.. after some more reading of fora, it seems that https://github.com/gerryjackson/forth2012-test-suite/ is the thing to clone.. it seems to be appreciated by the committee people. Seems to be organized pretty nicely in preliminary and incremental parts.. Doing, doing, done. Let's see.

Looks like very nice tests. I'm sure I'll be finding a lot of bugs and correcting a lot of nuances to make the tests pass :). At first, I'll need to actually fail on undefined words… knew this day was coming..

Note to self and others: If somebody was to create a new Jonesforth -like tutorial, I would much like its source to be organized according to Gerry Jackson's preliminary tests. Just in case somebody would be implementing their own Forth from scratch, like me here, and would like to be informed of a nice and useful test package from the first day of project :). It could be a faster and nicer route.. not sure if it would be as instructional as when you'd have to figure out your own tests while you go.. but surely easier, faster, and nicer.. preferrable if the time for studying is limited for any reason.

2020-07-21T08:53 - 2020-07-21T11:37 (2.5 hours; 53 total)

The old enemy of procrastination again.

I just wanted to know if there has been some archived discussion about whether or not to include the newline in the input buffer.. Gerry Jackson's "preliminary" test seems to run fine in my system, but its example output doesn't include a newline with SOURCE TYPE whereas my implementation does. My implementation includes each newline in the source buffer and types it.

While I just wanted to know that very small thing, I spent hours reading about Forth 2012 committee members, their companies, and their scientific articles about Forth and other things. It was inspiring but totally orthogonal to what I wanted to do today.

I need to focus!

2020-07-21T16:42 - 2020-07-21T21:05 (4 hours; 57 total)

Decided to just try some other Forths: Gforth (version 0.7.3 that comes in Fedora 32 packages) and the latest evaluation version of SwiftForth (3.9.1 for Linux).

Ok. These two seem to work a bit differently in interactive mode. And also when piping through stdin. File input gives the same response.

I don't have file input yet, and I'm not sure if I need nor want it. It is not part of the standard core anyway. I don't have a proper interactive mode either.. will be the last things to do, just in case I happen to want them..

Maybe I'll leave this note here as a backlog. If there will be file input at some point then newlines should not be part of SOURCE. Interactive mode.. well.. I'll check it out..

```
SwiftForth (interactive):

SOURCE . . ( 29 and newline ) 29 134587448  ok

Gforth (interactive):

SOURCE . . ( 29 and newline ) 29 94619682861704  ok

SwiftForth (with forth source file as argument):

[nieminen@localhost ~]$ files/hacking/Forth/SwiftForth/bin/linux/sf hmm.fs
29 -134340608
...
S" hmm.fs" INCLUDED 29 -134340608  ok


Gforth (with forth source file as argument):
[nieminen@localhost ~]$ gforth hmm.fs
29 94476809107256
...
S" hmm.fs" INCLUDED 29 94476809362216  ok
```

Conclusion: Ok, ok, I'll do it in the same way as Gforth and SwiftForth… Newline will not be part of the string given by SOURCE.

Done. Moving on, with perhaps a notion that interactive mode and INCLUDED would be nice to have at some point. I'd have them already if I just did the coding and not the reading of everything in the Internet.

2020-07-22T08:11 - 2020-07-22T10:10 (2 hours; 59 total)

Carefully studied the preliminary test code of G.J.'s and really verified that it passes *and* that it would record failures if there were any.

Next up: Go to actual tests of core words after the preliminary ones.

2020-07-22T10:11 - 2020-07-22T11:47 (1.5 hours; 60.5 total)

Gerry's test rack seems very good. Core tests running in no time - and crashing, naturally. I'm looking at so much debugging fun. Start the timer for this part of the run :). At the moment, no time to actually debug anything. Some other life happens, I hope with jogging..

2020-07-22T16:07 - 2020-07-22T16:25 (0 hours; 60.5 total)

Fixed my REPEAT. And studied carefully the specification of IF THEN ELSE BEGIN AGAIN WHILE … understanding more and more all the time.

Onwards, to the next issue reported by the tests.

2020-07-22T23:15 - 2020-07-23T01:35 (2 hours; 62.5 total)

Next problem is with DOES> and yes, oh, yes, indeed, I do remember that this was a difficult nut to crack. According to test, I didn't crack it. Read, read, think, think, … sleep is required at this point of time in order to do the necessary activities.

2020-07-23T01:35 - 2020-07-23T02:29 (1.5 hours; 64 total)

A very good night's sleep happened. But DOES> was still a very hard nut. In fact, I should say it "is" and not "was". My implementation looks very silly and I still don't understand how it relates to the text given by the standard. As of now, I leave all my earlier false turns and wandering thoughts in the comments. Today it seems that 7 more hours in total were spent with DOES> in addition to those done earlier. I will clean it up later… when I finally understand the beast. So far, it seems to pass the core tests, and I did learn new things especially from the WEIRD: test case..

So.. 7 hours (which is 3.4 percent of this whole project from the start!) were spent in debugging DOES> which is now better but still very suspect. And I don't understand it fully. I will certainly ask myself the question "What was the hardest thing to do?" and I think that DOES> is now a strong competitor of being the answer.

Tests on DOES> pass, so I'm rushing forwards to the next failing one. It seems to be with the number formats and conversions. Ok.. that's not difficult but it's tedious and really annoying. I'll have a good break before looking into it more carefully.

I'd love to celebrate a working DOES> but I'm not really happy with the result because I don't understand it fully, which leads me to anticipate that it is still somehow broken.

So it's not a celebrating break - more like coming to terms with very minimal progress kind of break..

2020-07-23T12:02 - 2020-07-23T17:30 (5.5 hours; 69.5 total)

Fixed <# # #> output. One line change that could be made in 10 minutes. Didn't really live up to the expectations of being a difficult fix :) which is nice after all that battle with DOES>.

Next up: Something wrong with >NUMBER as well..

Oh, I had made it single-precision by accident.. Then I tried a lazy fix that overflowed on double-cell numbers. I tried to find words in the standard that requires >NUMBER to work on double-cells, but didn't find a clear note. Tests seem to require double-cell >NUMBERing so I do it.. a little bit hard because of so much time since school times and doing algorithms with digits.. and I think I was a lazy boy and never learned them properly. Now I pay the price..

Done. Then one more thing.. dot-paren .(implemented to get no crashes or strangeness in the tests of core.fr

It's time to run the numbers at this point.. I got the "John Hayes test cases" passed at approximately 73 hours of this second project. It took some 23 hours from completion of first hack version to "Hayes-proof" version.

Let's see if there are any more ready-made tests for core.. oh, yes, there is the coreplustest.fth from Gerry Jackson. Try it right away (after reading through, of course). Nice, it has contributions from many people, and I think I recognize some of the test cases from the standard.. so I think I may have even "tested" them earlier (with less rigor and automation).

Oh, yeah! It's going to test corner cases of all those most difficult words that I've been fighting with all the time.. DOES>es and DOs and RECURSEs and all that. Oh what fun it is.

And.. boom.. it does make my current Forth crash with segfault in the fourth section of tests. With RECURSE, which I thought was one of the simpler ones. Sigh. Keep the project clocks running.. I'll look at it tomorrow after some sleep (a sleep of celebration because at least the Hayes tests passed and I may have learned a lot today; and there was some progress, even if slow).

2020-07-23T21:43 - 2020-07-24T01:45 (4.0 hours; 73.5 total)

Turns out my RECURSE wasn't causing the crash. Ehm.. I'm still causing mysterious crashes after any undefined word is used, and the coreplus tests use NIP and TUCK. Implemented those, and the test don't crash anymore. 15 minutes solution time. 30 minutes including this memorandum with a note to self: Resist the temptation of not doing proper error handling right now. You'll pay the price later, many times.

That said, I will do proper error handling for undefined words.. later. Students should do as I tell instead of how I do. Bad example is a good example when you know how bad it is and why.

Down to two incorrect results in coreplus tests. Looking good.

A real-life interruption happens for a while.

2020-07-24T08:01 - 2020-07-24T08:35 (0.5 hours; 74 total)

Three days were spent in other business / pleasure. How wonderful Rockfestari Naamat even if it was a stream event due to this COVID-19…

Back to this project… It turns out that the failing part of coreplus is using :NONAME from Core Extensions.. I guess I'll have to implement that to see the whole story about the tests…

Implemented :NONAME .. but this didn't fix all the test fails around those :NONAMEs. So, debugging further, but I'll commit my possibly working :NONAME before further changes..

2020-07-27T14:42 - 2020-07-27T16:36 (2 hours; 76 total)

My IMMEDIATE was immediate, and it should not be. It was tested by coreplus, and is now fine. IMMEDIATE is no longer immediate..

Now I get a pass from core and coreplus. Hmm… is it time to celebrate a "test-proof core system"?

2020-07-27T19:52 - 2020-07-27T20:59 (1 hour; 77 total)

Played around with coreexttest.fth from Gerry Jackson's tests. I don't think I want to celebrate too much until most of the Core Extension wordset is implemented and passing the tests.

This was not at all in my original plan, but it has changed. Reasons: (1) The core extensions seem to include some "obsolete" words that are likely to be found in older tutorials, textbooks, and legacy applications. Therefore, I think they are nice to know. (2) The extension seems to include also the opposite: words new to Forth 2012 that are supposed to supersede some of the old ones. I would like to call my system at least a lot like "Forth 2012", which means that I need to implement these new words. (3) A third category of words in the extensions seems to be words that are just absolutely useful, and used in so many tutorial examples of Forth I've seen so far.

It turns out that I have already decided to implement 21 of core-ext words (out of 49) in order to make it easy to work with the core.

So: Reset the first intermediate goal for this project:

- Implement the remaining words of both Core and Core Extension word sets of Forth 2012.
- Test using Gerry Jackson's tests up to coreexttest.fth
- Try to integrate the step "revise the whole thing according to current knowledge" into the above two.

Intermediate learning outcomes upon "8 ECTS"

I'll re-run my hour counts at this point:

- It took me some 214 hours to implement and test Core words of Forth 2012 and find out that it is not a satisfactory Forth system without implementing and testing also the Core Extension words.

How much is 214 hours of a learning exercise? It is approximately equal to 216 hours which translates to 8 ECTS credit points. In my own Faculty, we have a master level solo programming project of 8 ECTS called "Erikoistyö" (something like "Special project work").

Now that I tested it myself, I find out that 8 ECTS is a very good time to really go deeper into some topic. There is enough time in 216 hours to read, think, implement, test, and even procrastinate and have several periods of inefficient time and recover from long total breaks away from the project. As long as the inefficiency stays under control most of the time.

This is not to say that I learned to expect my students to be able to create and test a programming language implementation based on Internet tutorials and official specifications in an 8 ECTS project course. Most of them couldn't, because they don't yet possess the pre-requisite knowledge required to do so. But those who have taken some 15 ECTS of related pre-requisites - why not. Similarly, those who have taken 15 ECTS of pre-requisites in graphics should be able to really create some beautiful graphics code in 8 ECTS using textbooks, tutorials, and academic literature.

The bottom line is: I became certain that 8 ECTS is a long time and enough for great things to be done by any student under the condition that the student has done enough work in our curricular pre-requisite courses that teach the basics of "how stuff works" and "how to read manuals" in general.

This project of mine? It continues to its conclusion. Even though I think about these intermediate steps in terms of "Finnish university course lengths", the bigger picture is being drawn at the same time, and it will take as many hours as it happens to take. I have no time limits because this is a continuous hobby.

2020-07-28T14:10 - 2020-07-28T15:52 (1.5 hours; 78.5 total)

Onwards…

Implemented and tested <> U> UNUSED and WITHIN from core-ext.

2020-07-28T23:05 - 2020-07-29T00:35 (1.5 hours; 80 total)

Implemented and tested ?DO BUFFER: MARKER TO and VALUE from core-ext.

3 tests fail for ?DO but the bugs may be in my LOOP stuff from last year.

2020-07-2900:35 - 2020-07-29T04:13 (4 hours; 84 total)

Wow.. during the past 3 hours:

- Implemented and tested C" from core-ext .. as yet another copy-paste-modify-fix-later of earlier "-functions.
- Implemented SAVE-INPUT RESTORE-INPUT from core-ext .. found out I already had .R adapted from Jones.
- Implemented PAD and ERASE from core-ext.
- Implemented PARSE-NAME from core-ext.
- Copy-pasted reference implementations of DEFER DEFER! DEFER@ IS and ACTION-OF from core-ext
- Copy-pasted reference implementation of HOLDS from core-ext

I'll copy-paste a comment verbatim, to give a snapshot of today's feeling while rushing towards a complete core-ext implementation:

I thought these DEFER things would be difficult to implement.. Well, that was because I thought it was necessary to understand them and come up with the implementation… It turned out that DEFER DEFER! DEFER@ IS and ACTION-OF had complete reference implementation in the standard text. There is a catch as of now…

TODO: Learn and *understand* the rumoured beauty and usefulness of these DEFER things! That part remains undone in today's copy-paste excercise to get tests running.

Wow.. Missing are:

- [COMPILE] which is obsolescent and not even tested by current core-exttest.fth from Jackson
- CASE OF ENDOF and ENDCASE
- Debugging the 3 failing cases found in tests of ?DO

These will be frustrating, so leave to a later hour. These 3 hours were a surprise. I really thought it would take much longer to get even this close to full core-ext.

Quickly, commit and push these notes.

2020-07-29T11:29 - 2020-07-29T14:38 (3 hours; 87 total)

Wanting to do and not doing. Too much noise from the outside world. The old problem of inefficient hours again. An honest number is about 3 this time, I think.

2020-07-29T18:00 - 2020-07-29T21:00 (3 hours; 90 total)

deliberately fails tests without crashing. (Parses but doesn't do anything)

Situation: My system can run all the Core and Core Extension tests of Gerry Jackson's Forth 2012 test suite without crashing. 52 cases fail.

The crash-less test run is a win in itself. 27 failing cases are due to the deliberately non-functional S/". 18 failing cases are due to not yet implementing the CASE .. ENDCASE show at all. Rest of the fails are due to something wrong in the earlier implementations of :NONAME and ?DO or possibly something deeper that just surfaces in the tests for these couple of words.

Soon, after some sleep and stuff, I need to decide on the next battles:

- Struggle to make CASE etc in the obviously bad way I've built the earlier control structures?
- Refactor the old code, and go to learning mode (textbooks, other implementations, etc...)
- Accept some unfinished business in the core and already push onwards to the ultimate goal of making tiny audiovisual demoscene products? There will be many nasty steps on this passage too, which will have to be taken sooner or later. [as a later note, 2020-08-12, I'm happy that I did pursue this one. But I'm also very happy about the couple of additional exercises I decided to take on the way. In a way, I think it really doesn't matter what you do as long as you just do it and go forwards.. Makes any sense?]

Hmm... I'll sleep with this question in mind.

And I'll be happy that from now on I will never fall behind from at most 52 failing tests and 133/133 implemented words in Core and 45/49 in Core Extensions. These are very nice numbers to get some sleep with.

2020-07-30T00:39 - 2020-07-30T02:40 (2.5 hours; 92.5 total)

Ok, I just had to take a peek at the CASE .. ENDCASE story from the standard. Doesn't look very hard at all, from today's angle. I think I need to prove that point to myself before continuing with other business.

2020-07-30T02:40 - 2020-07-30T03:36 (1 hour; 93.5 total)

Now I was bitten by earlier laziness in an extraordinary depth. This just became a priority fix: There must be a sensible error message and preferably a crash with a bang when trying to compile an undefined word! I traced and debugged and verified so many assumptions before noticing that my bug was that I wrote

[,] instead of [']. The 1 hour job became a 4 hour job. Educational about many things, but it is a bit embarrassing that at the age of 41 I still have to be re-learning the same thing they told me in "programming for kids summer course" when I was 10: not to be lazy now because it will bite you later big time.

Nevertheless: boom, boom, I have a full Core with Core Extensions implemented and the number of failing tests just went down from 52 to 30. Turns out that the :NONAME RECURSE tests were using CASE and

problem with ?DO. Wow, actually it is only 3 fails with ?DO and the

Hunger just got bigger.. Next I'll use my brand new CASE to just make

with my new understanding. Any problem with ?DO will probably be easy to solve now.

I'll have to make it a bit more beautiful before breaking it with minimizers or integrating window things..

2020-07-30T23:40 - 2020-07-31T03:32 (4 hours; 97.5 total)

Towards the last 30 hours of this run

BTW: it is soon a very good time to start removing all that "old bullshit", i.e., things I have learned and the process with all its false turns. In order to preserve a possibility to take a later look into those learnings, it will be nice to make an annotated git tag at the point of no return to old bullshit… Quite soon, in project hours, I believe. [later, we know it was exactly at the point of 135 plus 135 hours, OF COURSE]

Hmm… troubles and troubles.. Note to self: must inform somebody that

sleep after all this debugging of not only own code but also that of test codes… continue later.. sleep.. now..

Remember to notify somebody of the case-insensitivity of tests. First check standard if FIND actually should be case-insensitive.. sort of remember it can be defined by a system, and a program has dependency if it uses lower case for standard words..

2020-07-31T23:21 - 2020-08-01T03:15 (4 hours; 101.5 total)

I just had to do it.. it took a bit.. ok, a *lot* more time than I expected. (The usual thing that happens). One reason was that once again I tried to code without being completely focused on the task. Also, I am always frustrated with all this character conversion stuff. Things I want to do as seldom as possible, no matter which platform or device. I just hate it.

Oh, the things you must do to fulfill a spec.. Comes with the trade, of course.

After all the trouble it is always a happy place: Only the 3 test fails due to ?DO are left. Numbers look good now. I want them perfect, so next up is debugging

?DO.

2020-08-01T19:21 - 2020-08-01T20:45 (1.5 hours; 103 total)

First the fix in ?DO. Was quite quickly done, after all.

2020-08-02T00:30 - 2020-08-02T01:35 (1 hour; 104 total)

Ehmm… Now my system actually crashes with a specific exit code when there is an undefined word in the input. It took one hour to fix this. I knew exactly 359 days ago that it should be fixed. I lost at least 10-20 hours of project time (10% of total!!), if not more, to

silently corrupt the stack. Or a [,] vs ['] or DROP2 vs 2DROP…

Why did I not fix this 359 days ago? Why did I not fix this 358 days ago? Why did I not this fix 357… you get the point… I try to become a better person.

How can I make my students learn this thing in their first years of learning programming from the beginning… With what kind of mind laser can I inscribe it in their heads: "Don't be lazy in places that are likely to mess you up later" … "Do the work now, so you don't have to do triple work in the long run" … "Fail fast, fail often, and make sure that any corrupt input data fails even faster and more often and ALWAYS IN A WAY THAT CAN BE IMMEDIATELY OBSERVED!!!" …

2020-08-03T10:20 - 2020-08-03T12:15 (2 hours; 106 total)

Spent some time reading the rest of the wordsets in Forth 2012. And some discussion threads about standard proposals… never mind..

Wow.. A lot of very useful and generic words are defined in the Optional Programming-tools word set. As an afterthought, some of them could be very useful to implement as first parts of the "bootsrapping" Forth code.

Hmm, yes.. but for learning, it might have been a good route to do things in a more difficult way at first. A hard way can be a fine way to remember by. Except when time is limited.

It looks as if the SEE word will be quite useful in examining how other Forths work.. for inspiration.

When refactoring, the "tools" wordset may give spoilers (and standard names) of useful factors.. CS-PICK CS-ROLL and the sort.

Some procrastination, yes, but also initiated the last steps necessary to get SDL2 and OpenGL hooked via hardcore-hacked hash strings. This will take some time.. Let's see if the 135 hours of this study project sequel will be enough. Learnings have been good, nevertheless. Good trip going on.

The proof-of-concept with triangle

2020-08-03T22:00 - 2020-08-04T03:50 (6 hours; 112 total)

Towards the goal. One inch at a time.. translating some hacks into Forth from last summer holidays' attempts made in C and asm. I think I'm getting the function names and call addresses OK, so the only thing missing is hashing the names into a call table and then actually figuring out a way to do the calls. Interesting times indeed.

2020-08-05T22:10 - 2020-08-05T23:55 (2 hours; 114 total)

This seems to be a night time project anyway.. All kinds of interaction and sauna stone assemblies and stuff in the daytimes. Tired but borderline capable of thought… Some straightforward copy-paste-modify from last summer's asm and C codes. The modify part is a bit hard because Forth is a little bit different logic with its stack orientation and stuff. Slow, but progress. Functions now somehow hashed to pointer array.

Yes yes.. approaching. I get an SDL window on screen and can read millisecond ticks from SDL. Nice start. I shall consider myself done with the proof-of-concept phase when there is the triangle and the beep. Soon-ish, I hope. I still have 18 hours before the magical 2 times 5 ECTS time frame has been used from the first lines of code.

2020-08-06T21:57 - 2020-08-07T01:11 (3 hours; 117 total)

Now it beeps and syncs nicely with draw. Audio mock-up is still hardcoded in asm, not yet audio from Forth side. Just a couple of more things to do in order to meet my original definition of what the proof-of-concept means:

- Enable pre-computation of audio track on Forth side and transmission of the track location to a "dumb playback" on asm side.
- Actually draw a triangle instead of 2D rectangle. With shading.

I think I will have to have some things from the optional Floating-Point word set and its extensions. Definitely not all of it.. just the necessary things for meaningful synth and graphics.. store, fetch, basic arithmetics, and sine and cosine .. that should do it. No fancy stuff. I may be able to pull this off in the 10 ECTS time limit.. if I just code instead of writing these notes and playing with the new beeper code. Focus! Focus!

2020-08-09T02:01 - 2020-08-09T03:40 (2 hours; 119 total)

That was the weekend that was supposed to be Assembly Summer and related activities. How strange that it didn't happen now. This COVID-19 year is very, very, strange. Well… there is time to actually prepare for future events. Onwards with this project. Try to focus.

… 5 hours passed… How was the focus?

I re-checked the AMD64 manuals about floating point instructions. And I did click on the Internet, too.. Four hours can go by very fast with a buzz in the head. Lots of nice readings about iterative function approximation algorithms and how floating point operations should be done in a proper compiler or math library these days. And I got all the more convinced that this project should use the good old x87 instructions because they give hardware implementation of all the necessary functions and a hardware floating point stack - all with very minimal code footprint.

I'll just check that I can safely delegate the Forth FP stack to x87 .. Yep, it seems to be fine to use the fp(0)..fp(7) stack of x87 and just delegate the operations to hardware.

Opcodes would be very short - but still at some point I'll probably want a shorter way to bootstrap the assembly side. I think something similar to actual Forth systems .. I'll have to examine them for inspiration. But I want maximum totality in size minimisation, so I'll probably be doing something really nasty if I can.

But, one step at a time. I'm ready to hook up the x87 floating point stack as quickly as I possibly can.

[Note added later on 2020-08-12: The reader should understand that it took me 5 hours of standing still and looking around before I could make the decision to start running towards the next decided goal. At this point of the project, all of these 5 hours of practically not moving did seem like a very long time. I understand that it could be frustrating to a young student to be basically "stalled" for 5 hours or even more in a task that is part of a course that only credits a pre-determined number of credit points.. But it is extremely necessary because otherwise there is no direction and any random direction is likely a wrong one.]

2020-08-09T23:03 - 2020-08-10T03:59 (5 hours 123 total)

The first few: F! F* F+ F- F/ F@ FALIGN FALIGNED

2020-08-10T11:20 - 2020-08-10T13:27 (2 hours 125 total)

Code, lunch, continue. Very disciplined for once..

2020-08-10T14:30 - 2020-08-10T15:33 (1 hour 126 total)

Code, put stones in the new sauna, hide thermostat cables, continue coding. Real life needs attention sometimes..

2020-08-10T17:35 - 2020-08-10T18:00 (0.5 hours 126.5 total)

Then more real life.. wait until it is night. Fortunately it is still mostly holidays and no early morning meetings. So few days left .. must go full speed now.

Implemented FCONSTANT FDEPTH FDROP FDUP FLITERAL FLOAT+ FLOATS

Updated comments about refactorings-to-be. Yep, the speed is good. Gotta remember: write less notes, more code now..

Implemented FLOOR FNEGATE FOVER FROT FROUND FSWAP

Phew.. The magical 10 ECTS credits is about to be full, too.

2020-08-10T23:45 - 2020-08-11T03:36 (4 hours 130.5 total)

Implemented FVARIABLE and then FCOS from float-ext to auditively test continuity of playback. . .

Oh, yes.. the playback.. of any audio pre-computed on the Forth-side!! Beep in this proof-of-concept.

It beeps. Now the *only* thing missing from the original proof-of-concept idea is a shaded triangle!!

Can I do the triangle in the 2.5 hours that I have before hitting the magical 135 hour maximum time?

2020-08-11T23:45 - 2020-08-12T01:55 (2 hours 132.5 total)

I was going to sleep, but then I made the triangle.

Only missing thing is to have that triangle shaded in a controllable color.

But *now* I *really* go to sleep. Holidays are over, at least when it comes to complete days. Scheduled meeting already tomorrow well before noon. . .

2020-08-12T02:05 - 2020-08-12T02:31 (0.5 hours 133 total)

Back to this as soon as I could after the work meeting and lunch.

Hmm, it was a max 15 minute job to shade using fixed pipeline glColor3fv.

My 135 has 2 hours remaining.. Can I also get custom shader compilation?

BOOOOOOM!!! It's there. I made it. It's committed and pushed in just a couple of seconds.. I'll do the aftermath and commenting later, because that's what I inevitably do. As of now - celebration:

Proof-of-concept meets its originally planned goals: It now beeps and shades an animated triangle synchronized to common beat! The contents of both audio and graphics is fully defined using Forth source code that works on top of my own system!! No assembly or any other language than Forth and GLSL is needed for audiovisual content creation anymore.

And this has been done within the magical time frame of 270 hours, i.e., 10 ECTS credit points worth of self-study time, starting from complete scratch. Peeeeerfect!! In the end, I could write this much of notes and still have a spare 10 minutes before reaching deadline. Oh, joy.

More analyses and aftermath later. Now the commit and push.

2020-08-12T13:50 - 2020-08-12T15:38 (<2 hours <135 total)

Aftermath / Self-assessment

I read the earlier notes and fixed some typos and language issues. It took 2 hours, which I didn't track nor record exactly. Quite a lot of notes, but it was an important part of this exercise to make these.

What remains to be done is the final self-assessment and conclusion of the past 135 hours. I tried to be brief and in line with the concluding note in my first 135 hour log in September 2019 (see the first diary file). I ended up doing quite a bit of work here, for which I count hours in the next part of the project.

Eleven months ago, in September 16th 2019, I had just finished my "First course in implementing Forth", totalling 135 hours of self-study. Then I set out to do the following items in my "Second course on Forth implementation". I managed to do the following planned items within the second 135 hours:

- "Implement the remaining few words of Forth 2012".

  Some 70 hours into this goal, I had to extend the goal as follows:

  - Implement the remaining words of both Core and Core Extension word sets of Forth 2012.
  - Test using Gerry Jackson's tests up to coreexttest.fth

  In the very end, I needed parts of the optional Floating Point words in order to make the graphics and sound proof-of-concept.

  Finally, I ended up with the following: complete Forth 2012 Core and Core Extensions with no fails in commonly used tests; two thirds of Floating-Point words.

  Thus, the original implementation goal was reached and exceeded by far.

- "Hook up SDL2 and OpenGL and make an actual proof-of-concept of demo. Window with triangle and beeping would suffice at this point".

  I have a window, triangle, and beeping. Original goal reached.

The following goal happened in a very unexpected way, and earlier than it was supposed to:

- "Then, it is VERY, VERY IMPORTANT to take a complete break from this project and spend my time on other interesting ones!"

  Extra full working year and COVID-19 made sure that a complete break from this project happened. It took close to a full calendar year to get the 135 hours done for this thing. And very, very, unfortunately there was no time for the other interesting projects either.

  Yet, I am very happy that I could finally do any hobby programming this year, and that I could focus on really getting progress in one project and not starting a dozen other interesting ones :). I commend myself for being able to focus on one single task.

The following goals were postponed to a further date:

- “Revise the whole thing according to current knowledge”.

  Some two weeks ago I made the final decision to completely drop this goal from this second part of my overall project. I’ll recapitulate my justifications for the decision.

  Firstly, I think that the revision will be better and faster after gathering even more knowledge than was available up to now.

  Also, I think the resulting system is now a better example of the steps in my learning process. I want this to be an example to young students who may not have yet experienced a similar trip themselves. I think that witnessing an intermediate situation in which their teacher is still completely clueless about some things should be reassuring. The pain of being clueless should be a tolerable part of learning.. I believe it is also necessary, and I want an unmasked example to be visible for the interested student.

  The 270 hour version consists of layers of incrementally added code of varying quality, some of it already dead and some to be killed as soon as possible - and pretty much every piece of thought behind them left visible as comments. Some of the thoughts are obviously wrong and some of them are conflicting with the one right next to it.

  Thus, the 270 hour version is probably not very useful for learning about Forth, because good thoughts are buried in a load of not so good ones, TODOs and FIXMEs. But I think that, from an educational perspective, it is all the more useful as a comparison to a future cleaned-up version for the same reason. Cleaning and straightening the code and especially the comments will necessarily destroy the ugly but insightful view of the gradual and constantly incomplete learning path that took place in the first 270 hours.

  I’ll tag this version so that an interested student can go and look around in my head - to see that it is supposed to be a mess most of the time when learning something completely new. It’s not always like “read this, watch that, do those exercises and you’ll be fine”. It’s not supposed to be, after introductory courses. it should get a bit ugly and painful, with no predescribed set of information.

- “Meanwhile, start reading Thinking Forth”.

  I think Brodie’s “Starting Forth” was more useful for these first steps and first 270 hours with Forth. “Thinking Forth” is definitely on my TODO-list for the next steps with this thing.

  So far, I've been pretty much "just following spec" and building standardized atoms. Now, with going to application and library building comes more opportunities for creativity, and it will probably be useful to understand more of the philosophy advertised to be found in Thinking Forth.

- The crazy dream is then to size-minimize this and make a 4k intro to Assembly Summer 2020 compo with the "techniques used" info field containing "Standard Forth 2012 running on my own interpreter". (== contribute smashing entries to Instanssi 2020 and Assembly Summer 2020 competitions…)

  Apparently the smashing entries did not yet happen this year. But of course Assembly was postponed due to COVID-19, so there would be a couple of weeks grace… But a few weeks might not be enough. Now that I've spent 270 hours building the system, I really want to put hours also into the entry and try to make it nice and demo-ish. I can't go with just a beep and a triangle… Moreover, I think it might be necessary to port this into Windows for easier running in a demoscene compo. Maybe; depending on compo rule details. These are thus more likely to be seen in 2021, unless I miraculously find spectacular amounts of free time before the new date of Assembly Summer in a couple of months. Probably won't happen because I need to attend to students and teaching starting next week.

  But 2021 is as OK for me.

What's my score?

I hate giving and receiving numerical grades, but when there are enough subcategories, it does give pointers as to which areas need improvement over others:

- Quantity of functionality implemented: 5/5
- Quality and elegance of implementation, avoidance of lazy hacks: 2/5

  (not a total miss, but can never be Good because of the damn

- Quality of learning, i.e., understanding the implementation: 2/5
- Creating and maintaining a plan, sticking to the plan: 5/5

  (Exemplary, in my own opinion. Always verifying plans and changing them if necessary)

- Time management between work, family and hobbies: 2/5
- Making realistic time estimates for project steps: 2/5
- Survival in spite of challenging circumstances: PASS apparently :)
- Overall: Let's say 3/5 (also the mean of above subscores). I think I did good but not spectacular.

What was the hardest thing to do:

- Obvious answer would be “understanding DOES> and perhaps POSTPONE”
- Real answer after reflection is: Scheduling, time management between life, work, family and hobbies… and things I really shouldn’t be doing at all. And facilitating the opportunities to focus on a task.

What would I do differently, if I could go back and change something:

- Try to be better focused on any task currently at hand. How? This is difficult, and a main thing to ponder for me.
- Do as you teach, man: I lost 10% of time because I didn’t do proper handling of faulty input on day 1, day 2, …, day 359 of the project. The fix ultimately took less than 1% of total time.

Main findings were:

- I think this 135 hours was pretty incremental on top of the earlier part. No new major discoveries. Maybe it was of special interest to see how much time and effort actually goes into picking up a project that has been untouched for many months. Importance of comments and backlog notes!
- Naturally all practice with AMD64, Linux, ELF, SDL2 and OpenGL and looking at their specs ever deepens the knowledge of the technologies. But it is all incremental and not eye-opening anymore.
- It is super important to emphasize the importance of handling or at least observing faulty inputs. It is super important to teach this to the young student, before it is too late. Keep this in mind when designing course exercises and evaluation criteria in the future…

Where do I want to go from here:

- Finally, the fun part begins: Now that it’s built, structured, and tested, I break and obfuscate it.. Size minimizations to the maximum extent! All kinds of wild ideas are just waiting to be tested in practice.
- The other fun part: An actual demoscene intro compo entry. The crazy dream itself.
- While minimizing, I want to keep it organized, and every obfuscation and size minimization trick must happen automatically so that source code remains standard and nice.
- Keep track of size minimization while trying things, so that any progress can be easily seen and quantified later.
- And now, read Thinking Forth and source codes of other Forth implementations to learn more of the Forth philosophy.

If you have made it this far, I salute you: Well done, or at least well read.. Next you may want to go learn something by yourself instead of peeking into my mind as it was in 2019-2020 while I was building my first Forth :). I hope these notes have been a useful example of how to set your goals, track your progress, and self-evaluate your old and new skills every now and then.

If you happen to be my student, here are some more pieces of advice:

- Use your brain and make your own decisions instead of just copying mine!
- For example, you will likely be better off spending more time in learning and doing instead of writing notes!

  I tried to make these notes exemplary and directed to a big audience which requires a lot of time and revising. As a student, your main task is to learn a skill and document it for only two persons: yourself and (secondarily) your teacher. So many more typos are allowed than when writing for the whole world :).
- Also, I think it will be beneficial to write in Finnish, if you are a Finnish student. It is likely that you are about to embark on a technology or programming course instead of an English language course. Thinking and writing may be easier in Finnish. If you are not Finnish or Swedish, you're out of luck - I'll need to understand your text, so you'll have to write in Finnish, Swedish, or English.
- When tracking time and tasks, you don't have to make it a super big deal. I encourage you to use automatic time-tracking tools, pen and paper, office software spreadsheet - whatever feels natural for you.
- Don't be too nervous about which hours to record. If you honestly attempted to use a couple of hours, but had no progress for some reason, just count in those hours and make a note about the possible reason why you had no progress. I tried to do exactly the same in my notes, too. It is just not possible nor expected to be running on all engines all the time. What is useful is to think about possible root causes of inefficiency in order to weed them out in the longer run.
- As a teacher, I'm expecting to read an honest account of what you tried to learn, and how it all went in your own realistic opinion. That will make me happy to reward the course credits that we agreed upon before you started. It is likely that I also need to see some source codes, version control repository, or other results of your project, but that's been individually agreed as well :). You are allowed to include in your project the hours you just spent examining these examples of a learning diary. I hope you recorded the starting time somehwere.

This learning diary continues in notes3.txt towards the academic working year of 2020-2021, the little time left for hobbies during it, and my self-studied "Third course in Forth: Application programming and extreme size minimisation tricks with demoscene intros in mind"

Part 3: "Third course in Forth: Application programming and extreme size minimisation tricks"

(Learning diary of a hobby project.)

I record hours spent in the "aftermath" of the previous part here.

Previous notes updated and grammar-checked to the best of my current ability.

2020-08-14T01:52 (3 hours)

Transition between part 2 and part 3

Revise the previous notes, once more.

2020-08-15T15:58 (3 hours 6 total)

Decide upon a license (X11 as of now). Further clean-ups of notes but not code, to make this a better example.

A side note: It has taken a lot of time to tidy all the notes at this point - almost to the point of being not fun anymore.

New #1 rule for myself: Calm down and write fewer notes. The example of planning and documenting has been established adequately in parts 1 and 2 already, and it is now time to set an example of doing instead of documenting in part 3!

2020-08-16T02:01 (2 hours 8 total)

One more hour of polishing and documentation.

2020-08-16T17:00 (1 hour 9 total)

Some more "tidying up" which means clarifying the labels given to parts of the mess. Nothing was deleted yet. But yes, a lot of deleting must happen right after this. Decide on the tag naming. Ok, next just do the tag, and start deleting stuff…

2020-08-18T00:15 (1 hour 10 total)

About to create the tag "POC-270h" without spending any more polishing wax.

There. Done. Pushed to github. Onwards, ASAP!

Detailed plan for the very next step:

- Separate implementation to files according to purpose - otherwise gonna drown in spaghetti sooner or later..
- Set up proper metrics for this part of the project: Size of executable of audiovisual demonstration would be the prime candidate?
- Kill your darling.. rid of files and comments that are not necessary anymore.

- Go through the optional Programming-Tools words, and implement useful ones. That's likely to pay back greatly in later steps.
- Then, re-iterate future plans.

Side note:

- Remember to call back to the Forth community.. tests probably should work with a case-sensitive system. Is it really OK to use NIP and TUCK and :NONAME in coreplus tests that are not yet testing core-ext that defines these words? Is CASE used in other tests before it is tested itself?

Now sleep, and then work, and then do the other important things in life… repeat… and then do the above when it is possible to return to this project.

Did I write "sleep".. yes, that is the first part of the current plan.

2020-08-19T23:41 (1.5 hours 11.5 total)

Yep, did sleep, and did work, and now to focus on main things:

- Delete, reorganize.

Yep. Started this pretty well, I think. Remember for next time: Just keep killing your darling. A lot of delete is best for this project right now.

2020-08-22T01:25 (1.5 hours 13 total)

Now that I have a full core system that passes all tests, I need some new metrics for the next stage.

From here on, I'll be interested in the kinds that are now automatically printed:

```
---------------------------
    Error Report
Word Set             Errors
---------------------------
Core                    0
Core extension          0

...

---------------------------
Implementation status:
  Implemented 133 core words out of 133
  Implemented 49 core-ext words out of 49
  Implemented 22 float words out of 31
  Implemented 3 float-ext words out of 48
---------------------------
Cleanliness status:
  #FIXMEs:  28   #TODOs:  52
---------------------------
```

```
Current gzipped sizes of ELF file, core system, proof-of-concept with gfx & sound:
3735 packtest.felf.gz
5786 packtest.core.gz
7524 packtest.poc.gz
```

Goals to measure: Keep implementation at 100%, test failures at 0, and start minimizing the size of zipped elements (the original driver of the whole hobby project). Meanwhile, it would be nice to bring down the number of unfinished parts, marked as fixmes and todos.

But now.. the work of the academic year begins, and it may be bye bye to these hobby projects for a while??

2020-08-24T09:21 (1 hour 14 total)

Done since figuring out what to do:

- Separate implementation snippets to files according to purpose
- Set up useful metrics for this part of the project
- Kill your darling.. rid of files that are not necessary anymore.
- Then, re-iterate future plans (see immediately below).

The obvious next step:

- Create a more minimal bootstrapping code, and refactor the basic atoms written in assembler. I will be spending some nice hours with felf.asm

  NOTE: it is important to go through the optional Programming-Tools words, and implement useful ones, perhaps early in the bootstrapping process. That's likely to pay back greatly in later steps (?).

- Keep killing the darlings.. rid of dead code and comments that are not necessary anymore. Especially old, puzzled, wrong understandings.. now that I've learned more, some of them seem just silly.
- Prepare for more ultimate size minimisations, i.e., at least:

  1. automatic and total removal of words that were never used in a "publishable final production"; think tracking word execution with some RESET-USED-WORDS Run-my-application USED-WORDS etc…
  2. automatic obfuscation of word names to few characters length, something like USED-WORDS >OBFUSCATED-NEW-WORDS
  3. automatic inlining of constants,
  4. specifically inlining hashed function names for SDL and GL calls instead of original symbol names.

  These ultimate things will come later, I'm sure.. I want to read Thinking Forth and get yet some more experience with Forth programming before

> attempting these. My guess is that they would be most elegant as some "Cross-compiling, self-obfuscating Forth on top of my standard Forth".. interesting stuff, but I'm not there yet.
>
> Nevertheless, I must keep thinking about these goals when making any decisions, in order to avoid duplication of effort or massive need of re-writing parts later. Hobby project hours are limited enough as they are…

That's it.. Ready to start with minimizing the bootstrap! When I have the time..

2020-08-25T12:25 (2.5 hours 16.5 total)

Reading the NASM manuals, in search of possible neat macro tricks.

2020-08-30T23:24 (2.5 hours 19 total)

The Winter Has Come. No more time unless I really, really try. And then it is sparse when there is any.

Quickly commit, and try to do at least something that looks like progress to some direction.

2020-09-07T22:43 (0 hours 19 total)

Yep - the obfuscation code is gonna be an awesome and unintelligible regex hack soon. But the original application code remains clear which is important. Babystep, but still a step forward. Too little time.. must sleep, work, sleep, repeat.

2020-09-07T23:40 (1 hour 20 total)

Furthered the obfuscation code

2020-09-12T10:00 (1 hour 21 total)

Phew.. 2 hours of nice Sunday coding time. Not much this week, but I'll have to do with the resources available.

I made a new bootstrap implementation that stores only densely packed counted strings as "name, code, name, code". And there's also initial ideas of omitting and renaming words (the obfuscator).

Lots of work ahead, so I'm expecting a long delay in calendar time from now on.

2020-09-13T14:31 (2 hours 23 total)

Next up:

- Use the new begin_bootstrap_code for everything that is possible.
- Make the assembly side as simple as possible, for example no need for $hex #dec %binary or other things that can be made on the Forth side by re-defining words. (The final minified executables won't need much parsing at all, so can revert to the assembly ones)

Surprising evening hours.. tired and unfocused, but I did manage to move some things into the new format. Packed executable size is clearly getting smaller by using the new bootstrap, even without obfuscation or removals. That looks promising.

2020-09-15T22:00 (1.5 hours 24.5 total)

Continue with the current sub-goal.

Note: It was “3527 packtest.felf.gz” before [var_*DP*] -> rdi

Then I continued trying to get rid of var_*DP* and use RDI instead. But it is a bit messy. And I was a bit tired in this session after a week of work. Good thing that there were some hours of opportunity, though, even if I misused these.

2020-09-19T01:58 (3.5 hours 28 total)

It was a bit more work and debugging than anticipated, but I got rid of *DP* in favor of using the RDI register as the data-space pointer in all parts of code.

New measure: 3442 packtest.felf.gz

So, about 90 bytes size saving from using RDI instead of a memory location.

2020-09-19T18:50 (2 hours 30 total)

Absolute addresses pack better. +100 bytes in executable, but -150 bytes in package.

New measure: 3289 packtest.felf.gz

Moved some more words to the new bootstrap method. Savings could be something like 7 bytes per word, for words with very little machine code. I like the way this is going.

New measure: 3220 packtest.felf.gz

2020-09-19T23:30 (2 hours 32 total)

More words to the new bootstrap method (double cell integer words).

New measure: 3198 packtest.felf.gz

2020-09-20T00:55 (1 hour 33 total)

And yet more.

New measure: 3113 packtest.felf.gz

2020-09-20T02:02 (1 hour 34 total)

And yet more. Almost everything moved that can be moved without also rethinking the whole implementation strategy. I’m sensing some slow and tedious times ahead…

New measure: 3074 packtest.felf.gz

2020-09-20T12:40 (2 hours 36 total)

Next up:

- Do something about the hack with calling words from others! Spend a few hours on this to make it nice. Maybe some usual functions to be called internally?
- Try to get rid of all defword's! Implement purely in Forth if need be. Make the assembly side pure assembly and not a

  mix. (pedagogically justified in Jonesforth but not practical in mine).
- Sort out the dictionary word creation.. So many words very similar now. Is it a problem, though, because zipping handles repetitive copy-paste-codes quite well?

First moved EVALUATE and ABORT from a defword to strap.fth. Things will probably get slow now..

2020-09-20T22:50 (1 hour 37 total)

One minute here, one there, unfocused, during today. I count them as probably an hour of futile efforts.

2020-09-21T23:59 (1 hour 38 total)

Moved more stuff to the new bootstrap method..

3015 packtest.felf.gz 5097 packtest.core.gz

(Start to record also the size of the Forth side of the bootstrap; things probably start to move there from the assembly side, so there will be some trade-offs to consider, maybe…)

Did some more of the easy ones. It's a surprise there are still easy ones. But soon there will be the difficult ones left (also, they are the ones I hacked together in a rush earlier).

2964 packtest.felf.gz 5042 packtest.core.gz

Oh, but this was a nice morning hour before starting to work. It's COVID-19 distancing, so distance to work is only 0 meters. Strange times. Switching context..

2020-09-22T08:30 (1 hour 39 total)

Reading the gforth manual and some Forth websites to get some different points of view. Looks like a nicely supported wild west. I want to stay as close to the standard as possible, but I also want to find extreme size tweaks. Nothing very useful found on this search.

2020-09-26T00:52 (2 hours 41 total)

Some surprise hours possible again. Moved more stuff to new method and also from assembly to Forth stage of bootstrap.

New measures:

2817 packtest.felf.gz 5000 packtest.core.gz

And after observing that even more could yet be moved "easily" before it gets difficult:

2801 packtest.felf.gz 4985 packtest.core.gz

2020-09-27T02:21 (3 hours 44 total)

Moved also the platform library calling codes to new bootstrap method.

Tweak CREATE CONSTANT : and :NONAME to become almost the same code. Common factors are quite visible already. Next up: proper factoring.. should make them all short and tidy.

2778 packtest.felf.gz 4962 packtest.core.gz

2020-09-27T23:30 (1 hour 45 total)

Disentangled the messy hack of calling words from assembly. Just call using call instruction. This was also part of sorting out CREATE and friends.

2728 packtest.felf.gz 4912 packtest.core.gz

2020-09-28T00:24 (1 hour 46 total)

Okay, this turned out to be a quite nice weekend.. Relaxing, not thinking about work, and surprisingly many hours of hobby programming time. Nice. Now, must sleep and dive into the working week.

2020-09-28T01:23 (1 hour 47 total)

Ahaa. Then I learned that CONSTANT was just : CONSTANT CREATE , DOES> @ ; so I get shorter everything:

2702 packtest.felf.gz 4894 packtest.core.gz

Due to my unoptimized engine, this is a mind-bendingly inefficient way to use constant values :D . . . but I have decided on the metrics to use, and the packed size metric rules over everything else.

Some 300 bytes smaller after this weekend, at the same time improving readability and keeping full functionality. Ok, Only some 150 bytes in the complete system, but still.. Only improvements without need for compromises. Wow.

Next up:

- Try to fix remaining issues with CREATE : and :NONAME
- Try to integrate the new factor new2dict into the main bootstrap loop
- See earlier "next up" because I may have been distracted..

Wow, time flies when it's fun and entertaining and playlike. NOW, must really sleep and dive into the working week.

2020-09-28T02:31 (1 hour 48 total)

Now. To sleep. Go. To. Sleep.

2020-09-28T02:57 (0.5 hours 48.5 total)

Several wrong turns and twists, but then likely some good ones, too. Got rid of the original defword completely, so could move a lot of stuff to the more compact storage format.

2621 packtest.felf.gz 4813 packtest.core.gz

2020-09-30T01:13 (3.5 hours 52 total)

2611 packtest.felf.gz 4806 packtest.core.gz

Some more things and tunings. Spent some time reading and clarifying old comments - at the same time thinking about further actions.

2594 packtest.felf.gz 4785 packtest.core.gz 6591 packtest.poc.gz

2020-10-03T00:30 (1 hours 53 total)

Almost sleeping.. not very efficient. Some bytes shaved by re-organizing. Clarity improved at the same time, so win without lose.

2586 packtest.felf.gz 4767 packtest.core.gz 6575 packtest.poc.gz

2020-10-03T01:40 (1 hours 54 total)

Moved SDL audiospec to Forth side. Minimal audio playback code remains in asm.

2517 packtest.felf.gz 4699 packtest.core.gz 6564 packtest.poc.gz

Moved CREATE to Forth side in an attempt to make the assembly side as small as possible. POC gets larger overall, but I have a hunch that this will be compensated after the thing gets more uniform and the minification kicks in.

2488 packtest.felf.gz 4704 packtest.core.gz 6564 packtest.poc.gz

Yet another attempt to touch this project while being interrupted by the real world all the time. I count two hours of approximate "efficient time" even though I had some separate minutes all over the weekend. I estimate that I could have done at least the same stuff if I had had two hours uninterrupted.

2020-10-04T00:30 (2 hours 56 total)

Next up:

- Rid of WORD and use PARSE-NAME instead?
- Finally, check the previous "next ups" if they have been dealt with..

How do the existing Forths PARSE-NAME … I still haven't gone to their sources. It is a quest I must accomplish at some point. Now I do some reverse-engineering::

: tp PARSE-NAME . . ;

Gforth considers 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x20 whitespace. SwiftForth evaluation version seems to have a similar PARSE-WORD but not PARSE-NAME … PARSE-WORD seems to recognize strictly 0x20 and nothing else as whitespace. This is a bit strange…

So… what shall I do in my own system? Aha.. I'll lean to Ertl's 2017 comment 'For the meaning of space delimiters, see 3.4.1.1 and 11.3.5. The same meaning is intended for "white space".' So, because, with files, control characters shall be treated as space, I find it natural to define the same for all parsing. Further, I decide to pick up the simple interpretation of the current reference implementation of PARSE-NAME: any character less than 0x21 is whitespace for me…

Simple. Immediate game plan is:

- Factor _PARSE-WSS (WSS=whitespace separated) that performs the standard function of PARSE-NAME (internal factor until I learn whether it is factually interchangeable with the parsing that should take place in INTERPRET, including numbers…).
- Implement _PARSE-WSS in machine code because it is essential for also the INTERPRET
- Implement, in machine code, non-standard _FIND-STR to work like find but for "addr u"-strings.
- Re-work all internal code to use _PARSE-WSS and _FIND-STR. This operation should make the code cleaner, leaner, and possibly smaller, too.
- Meanwhile, possibly implement POSTPONE in machine code.. It could enable the Forth-side bootstrap to skip some annoying baby steps in the beginning. Now a good time to peek into the source codes of other Forths, in order to settle my uncertainty about POSTPONE vs ' COMPILE, ?
- Implement the standard PARSE-NAME and FIND on top of the new internal factors. Revise the Forth-side which should resultingly be cleaner and leaner, too.

That will be a relatively large update, so expect to use a good number of hours. And then, because I've been postponing this too far:

- Go through earlier "next up" -plans to see if this project is still on track..

2020-10-05T08:35 (2 hours 58 total)

Halfway through the previous set of changes.. in only two hours. A little bit disappointing intermediate code size measures. But I'll go through with it and see if it remains a bad idea after all..

Some nasty debugging of non-standard things I already forgot I had used and depended on. Should learn to be safe, not sorry…

2020-10-06T01:00 (3 hours 61 total)

Nice coding night.. a quiet world around but still not very efficient because of being tired. Not enough sleep. Must go to work. All this work and no play…

2020-10-06T08:05 (1 hour 62 total)

Yep. Banished all uses of WORD in my own internals. WORD is now defined in a late stage of Forth-side bootstrap as "legacy support". Implementation is noisy beginner-work but passes tests; I'm satisfied.

Size didn't improve much, but code got leaner.

2479 packtest.felf.gz 4824 packtest.core.gz 6684 packtest.poc.gz

2020-10-10T12:06 (1 hour 63 total)

Reading and cleaning old code; planning for next steps. Trying to bend the brain around this hobby again after an intensive week at work. It's hard work to get out of work, mentally :). I admit, I may have a bit of a tendency to workaholism… Fight it, fight it.. well done today.

2020-10-10T13:39 (2 hours 65 total)

What is up in the mid-way cafe?

All right.. I did finally get to checking the previous "next up" plans. It is now a proper and symmetric time to review and revise plans, since I'm in the middle of the third "standard-sized course" of 135 hours or 5 ECTS credits.

Wow, a lot of the planned things have been done in only 60 hours. It looks like a long time, but basically it converts to only 1.5 weeks of full-time working time. Not so much, viewed through that lens.

First of all:

- I'm now clear of earlier "next up" items that are not repeated below.

Items not yet done (from earlier notes):

- Make the assembly side as simple as possible, for example no need for $hex #dec %binary 'c' or other things that can be made on the Forth side by re-defining words. (The final minified executables won't need much parsing at all, so can revert to the assembly ones)

  (went a long way into this direction by 2020-10-26, but there is much more to be done; on 2020-10-28 I think I've done all that I can do before the actual minification takes place. The assembly will get leaner when the syscalls and possibly number parsing go away during scaffolding with the obfuscation code. Those are to be done next as a priority!)

- Go through the optional Programming-Tools words, and implement useful ones. That's likely to pay back greatly in later steps.

  (went through these by 2020-10-26; greatest outcome was the revamp of control flow structures using AHEAD CS-PICK CS-ROLL and much easier debugging by finally implementing .S; learnings: implement these as early as possible, when creating your next Forth system ;D, in order to speed up the process in the long run)

- Call back to community about details in tests (added an issue in github 2020-11-04)
- Should remember to go towards the subitems laid out on 2020-08-25.

  (doing this all the time; becomes a permanent reminder until completed)

- Read Thinking Forth, and look at the sources of other Forths (look for nuances in POSTPONE vs ' COMPILE, and such things of interest)

  (started looking at gforth a bit in September; Thinking Forth yet to be started, probably in the next part of project)

- Keep killing the darlings, all the time.. (a permanent reminder rather than a specific item..)

Soon-ish to happen (stepwise re-iteration of the top priority items above, and two new items of importance at this stage):

- POSTPONE in asm (done 2020-10-17)
- Revisit INTERPRET (done 2020-10-26)
- Scaffold the obfuscation and reading bootstrap code from exe image. (done, tentatively, by 2020-11-04)
- Terminal input would move to a late Forth-strap phase as a result?

But, alas, another week in the working paradise.

2020-10-13T08:22 (2 hours 67 total)

Revisit plans and specify needed changes in the source.

2020-10-14T08:06 (1 hour 68 total)

Step-by-step, according to plan.. Adding done-notes to the above to-do.

2020-10-17T18:47 (1 hour 69 total)

Step-by-step, according to plan.. Adding done-notes to the above to-do.

2020-10-18T01:44 (2 hours 71 total)

2020-10-18T15:12 (1 hour 72 total)

Thinking about thinkos in some assembly conditionals.. Trying to wrap my head around the legacy : [COMPILE] POSTPONE POSTPONE ; IMMEDIATE

It is very, very frustrating that I have to still dig deeper into the basic concepts of compile semantics vs. other semantic modes.. When, oh when, will it become clear.

2020-10-20T08:54 (1 hour 73 total)

Figured it out, at least momentarily. That legacy definition from 1989 didn't work in gforth either, so I wonder no more whether it should work in mine. Treat it as legacy, trust the current tests, and just move on, according to the roadmap detailed above.

2020-10-21T08:30 (1 hour 74 total)

Tidying up the Forth-side bootstrap in preparation for re-working INTERPRET.

2020-10-21T16:23 (1 hour 75 total)

Size measures before revising control flow structures' code:

2520 packtest.felf.gz 4804 packtest.core.gz 6664 packtest.poc.gz

(I'm refactoring the control flow words once and for all, basically as part of preparing for INTERPRET..)

At the moment the first CS-PICK test crashes, so I need to debug before using toolstest. Must go to work for a day again..

2020-10-22T08:25 (2 hours 77 total)

Revised the 8 building blocks of control flow words using origs and dests.

2020-10-23T08:02 (2 hours 79 total)

Revised some of the standard control words to use the building blocks. New measures:

2520 packtest.felf.gz 4762 packtest.core.gz 6650 packtest.poc.gz

Next, try to battle the DO loop stuff with current understanding. (Not even really necessary for the INTERPRET thing underway, but such a nice exercise that I want to take it up at this point.)

2020-10-24T00:48 (1 hour 80 total)

Yep… not much of a battle, when you're equipped with great new understanding. Much cleaner and leaner code, and resultingly smaller too:

2531 packtest.felf.gz 4705 packtest.core.gz 6601 packtest.poc.gz

2020-10-24T19:10 (2 hours 82 total)

Further tidying makes it quite beautiful.

2540 packtest.felf.gz 4700 packtest.core.gz 6597 packtest.poc.gz

Outcome is good. What about efficiency? I did what I could in some 5-10 minute chunks over a whole Saturday. I estimate that 1-2 hours could have been enough for both thinking and doing the changes of today, but it was spread over 10 hours. Randomly suppose I should count 3 hours for this project, knowing that there was 1 hour of efficient time but I need to be "punished" for all this inefficiency in the total number.

2020-10-25T01:00 (3 hours 85 total)

Sizes after refactoring the string literal code:

2540 packtest.felf.gz 4688 packtest.core.gz 6581 packtest.poc.gz

Reduced number of lines in code. Much more readable now. Win.

And.. finally, I implemented the actual INTERPRET in Forth and got rid of BASE and anything but decimal literals on the machine code side.

Current sizes:

2481 packtest.felf.gz 4808 packtest.core.gz 6693 packtest.poc.gz

I got a hit of 120 bytes in the complete Forth 2012 system and 110 bytes in audiovisual proof-of-concept. But that doesn't matter because I anticipate that none of that code will be present in an obfuscated production executable. So, instead I win at least 60 bytes that is never coming back.

And that was my goal for this weekend. Goal achieved, but it was really difficult to find the time and commitment to be both a good husband and a good nerd. I need to take some time off from this project at some point and enjoy some non-nerdy times. Fortunately, the working week begins in a couple of hours, so at least I won't be coming back to these codes for a couple of days.

But then the question of how many hours should I say I spent on this project today. Basically 14 hours of wanting to do it, but at the same time doing completely different things. Maybe 3 hours focused on the keyboard and moving the beast forwards. So something between 3-14 should be estimated. Let us say I could have done this bit in one complete working day, i.e., 7 hours focused time.

2020-10-26T01:21 (7 hours 92 total)

Preparations to super-minimize the assembly-side of the bootstrap and create the obfuscation scaffolding. I think I'm on the right path, or at least the path that I have chosen :).

Now, off to work for another Monday.

2020-10-26T08:33 (1 hour 93 total)

Finally, INTERPRET no more exists until late in the bootstrap when it is built in Forth. Instead, there remains a bare-bones hack of a pre-interpret loop in the machine language side.

Notable decrease in the packed exe size:

2414 packtest.felf.gz 4762 packtest.core.gz 6647 packtest.poc.gz

Put : and :NONAME in the current bootstrap method. Getting smaller and smaller, without losing any functionality:

2397 packtest.felf.gz 4745 packtest.core.gz 6626 packtest.poc.gz

Some micro-optimisations and tidying:

2385 packtest.felf.gz 4727 packtest.core.gz 6614 packtest.poc.gz

Major breakthrough - rid of the whole defcode macro. There is now only one way to bootstrap:

2352 packtest.felf.gz 4703 packtest.core.gz 6587 packtest.poc.gz

A very wild idea: Now that I made the wonderful xt array, could I go back to the idea of bulding a "QUIT"-like structure from only dictionary words that are uniformly bootstrapped!? Because now they just became byte-long entities each. Could I make a word like BS-ARRAY-EXECUTE (u – i*x) and then go uniformly all the way.. In exe there would be the current bootstrap code and then just "boom" it's there.

2020-10-27T01:30 (7 hours 100 total)

Discard the wild idea of last night. I spent a good few hours considering it, and it would be troublesome. Perhaps possible but too much work. I'm already 27 hours (1 credit point) short of this part of the project. This goes into a backlog or perhaps some later project.

My understanding of threaded execution Forth got 3 hours deeper, though, which is nice.

I'm progressing well, albeit a little slow compared to the original goal of this 5 ECTS part. I think I'll have to postpone most of the "application programming". The extreme size minification things I may still have time for. Hmm… but then again - what is application programming in Forth? I think I may have gained some skills in that respect because application programming is not far from system programming because of how things work. Actual proof by showing an application must wait for the next part.

2020-10-27T01:30 (3 hours 103 total)

This weekend has not much to report. Being in real life, or something. Nice, of course.

2020-11-02T01:30 (3 hours 106 total)

Very difficult to find focused time. Very difficult. Last 5 hours feel like I could have done it all in just 1 hour if there were no distractions.

Well, I think I've gotten somewhere with the obfuscation / minification ideas, but that somewhere is rather disappointingly little forward.

2020-11-02T11:22 (2 hours 108 total)

Now, I can follow a new measure which is: The zipped size of an obfuscated executable ELF binary that actually runs the graphics proof-of-concept, containing everything!

2344 packtest.felf.gz 4691 packtest.core.gz 6577 packtest.poc.gz 6688 packtest.obfpoc.gz

Now this is fun. The code is a killer hack and must be mostly redone, but the current outcome is really promising. Going to work now… trying to stay off this project to get some TODOs to DONE state for the uni too.

Brief pointers for next actions in here:

- I need two passes in my obfuscator - of course, because I need to analyse which word definitions end up being used and which ones can be omitted altogether. I want automatic omission. Maybe even three passes (?): 1. Analyze. 2. Comment out unused pieces so everything can be revisited/debugged at this stage. 3. Dumb removal of comments.
- Fastest wins will surely come by focusing on constants and long human-readable names.

2020-11-03T08:30 (1 hour 109 total)

Only one more thing from earlier next up list remains, so raise its priority:

- Terminal input would move to a late Forth-strap phase as a result?

Now, to work. These morning hours have been very productive all-in-all.

2020-11-04T09:34 (1.5 hours 110.5 total)

Ok, so I've done all the earlier next-ups. Next up:

- implement WORDS and SEE and DUMP from tools. (lot of tools implemented already by 2020-11-16; SEE needs more time than available now)
- late entering to a stdin-reading loop, optional and likely not required for size-minified exes. (done, done, done..)
- Make instrumentation for the obfuscation: what are the words actually used by a program? This question can only be answered by witnessing the actual execution. It can't be reliably guessed heuristically because of all kinds of possibilities and possible ticking of xt's and so on… This is important in order to really decide on words that are omitted from tiny exes. Idea: USECOUNT-RESET ' demo EXECUTE USECOUNT-SHOW and then you get to know exactly the words that were used. Others can be omitted by the minifying obfuscator. (initial quick-and-dirty idea done by

2020-11-16; more elaborate attempts will have to wait for a much longer time, perhaps indefinitely)

2020-11-08T02:34 (1.5 hours 112 total)

The nice morning hour again. Implemented WORDS and prepared for instrumentation and stuff. Now off to work. Life goes on… not so many hours for hobbies.

2020-11-09T08:41 (1 hour 113 total)

Implemented some survival versions of the rest of the tool words.

2020-11-10T10:15 (1 hour 114 total)

[IF] [THEN] [ELSE] and COMPARE vomited out to code, mostly from reference implementation, but not tested.

2020-11-10T11:30 (1 hour 115 total)

Tested [IF] [THEN] [ELSE] and COMPARE and [DEFINED] and [UNDEFINED] using Gerry Jackson's test suite.

2020-11-15T00:30 (1 hour 116 total)

Last 20 hours of the third episode

Revise goal: Postpone the application programming. Focus on getting the size minification and obfuscation working. Don't even try to get any actual application code done. Try to enable a neat transition to application programming after this 5 ECTS part which is due in less than 20 hours as of today.

Yep.. Trying to count each word execution one by one. Hehe.. The audio pre-computation loop takes some time now.. Perhaps not a final solution after all. But was quick and dirty, and certainly never misses a single use of a word. Can be used for profiling, too. What alternatives are there? Recursive analysis that would do something like SEE for every SEEn word, and count the number of times that each one was SEEn during the recursion. That would be a logical choice.. but probably laborious because my current implementation mixes code and data making it impossible to SEE clearly as of yet, I think.

Maybe I stick to today's quick and dirty counter for the first proof-of-concept. Details to be revised later if there is still interest.

Pretty much the final next-up that fits the current 135 hours (minus final documentation and assessment which I don't want to leak into the next one this time):

- Finalize a crude minifier that (1) leaves out all unused definitions and (2) minimizes names of all used definitions (3) handles integer literals in binary form. (items 2 and 3 got something like "finished but not polished" already by 2020-11-20, quite quickly and painlessly after all; automatization of item

1 requires more thought and thus it must be postponed to the future; means of manual tweaking to achieve item 1 is already in place by 2020-11-20).

- Start wrapping up this part at 130 hours, and spend no more than 5 hours for assessments and documentation. (started this at hour 131, so trying to use no more than 4 hours to the wrap-up)
- (even this could be optional in the current 135 project!) Make entering to an stdin-reading loop optional and not required for size-minified exes. (for the most part, this was done by 2020-11-20, too, effectively resulting from the items 1-3 above; the thing about being "not polished" means that most of this nice functionality is now quite hidden in a maze of twisty little passages of placeholder-like crappy code… sorting it out is one top priority in near future.)

2020-11-16T10:30 (4 hours 120 total)

Some more quite crude hacks in the minifier. I'm gonna have to think quite a bit about the minification. Current version of the beep-triangle POC now fits in less than 6k and it still comprises the complete Forth system and several additional parts:

5820 Nov 17 17:22 obtest.gz

The gravity of automatically leaving out unused definitions is now becoming more apparent. Think, think, think, … Some of this will likely spill over to the next part of my project.

2020-11-17T18:00 (2 hours 122 total)

Trying to remove just two tiny words and mind the size in various other ways. Notable improvement in size (also found that I had this in my measures already, under different name):

5752 packtest.obfpoc.gz

2020-11-17T23:16 (1 hour 123 total)

Omitting KEY from the assembler side and the dependent ACCEPT from Forth-side, get savings of some 35 bytes - and a proof-of-concept of automatically omitting unused words from both sides of the bootstrap.

5715 packtest.obfpoc.gz

Yep, this is what I realised earlier today.. I have to do this one very small step at a time. Quickly commit, and continue playing with the new toy (no… remember the procrastination monkey… but I wanna play, and there must be fun in computing…)

2020-11-18T00:45 (1.5 hours 124.5 total)

So much fun, and witnessing a rapid decrease in the size that matters:

2360 packtest.felf.gz 4706 packtest.core.gz 6907 packtest.poc.gz 5400 packtest.obfpoc.gz

Yet, an increase in hours for this project that is coming to its next turning point very soon. Also too many hours into the night. Now, I go to sleep and to work..

2020-11-18T01:20 (0.5 hours 125 total)

Just quickly commit and push the minor changes made while playing before going to sleep. Now, off to work, before the playground gets too much attention.

2020-11-18T08:08 (0 hours 125 total)

Trying to figure out how to have numbers in binary format, and start getting rid of the complex code of % # $ ‘c’ interpretation - which is totally unnecessary for my goal of demoscene intros.

2020-11-19T08:28 (1 hour 126 total)

Some steps made towards better obfuscation. Now I removed the real INTERPRET loop and QUIT totally. Obfuscated size is getting smaller, and the increase in the bootstrap assembly part should be temporary (until I do figure out how the numbers get stored and remove the need for syscalls altogether.. and stuff):

2419 packtest.felf.gz 4784 packtest.core.gz 6976 packtest.poc.gz 5126 packtest.obfpoc.gz

Then I continued trying to figure out how to have numbers in binary format..

2020-11-19T19:47 (3 hours 129 total)

Phew… then I did figure it out! Great size reduction by just binary representation, even without removing the code that parses numbers:

2419 packtest.felf.gz 4784 packtest.core.gz 6976 packtest.poc.gz 5057 packtest.obfpoc.gz

And that’s it then. I want to wrap this part up with self-assessment and all that, before going into the next one. So… feature freeze, and just look back for a few hours, minimal tidy up and then onwards to the next episode.

2020-11-20T00:05 (2 hours 131 total)

Read through the notes and prepared the self-assessment below by copy-paste-shorten. Revised language of the notes to the best of my current capability.

Then I just had to do **one** more improvement of the obfuscator, i.e., hash the function names so there remains no long symbols anymore after obfuscation. Easy removal of 50 bytes from the obfuscated POC and I get my final measures promising with a figure starting with a “4”.

2419 packtest.felf.gz 4784 packtest.core.gz 6988 packtest.poc.gz 4999 packtest.obfpoc.gz

2020-11-21T16:15 (3 hours 134 total)

Wrapping up, and moving on

Self-assessment

How did I do?

I'll start by looking at what I was supposed to do:

- Work under the title "Third course in Forth: Application programming and extreme size minimisation tricks"

  Assessment: I didn't get to the application programming part. Yet, I think I learned a lot of things about Forth that will benefit application programming later on. I'd grade it 2/5 to signify potential yet undemonstrated or partly hidden success. Extreme size minimisation tricks went smoothly and simply couldn't have gone further, so 5/5 for that part.

- Work under the New #1 rule for myself: Calm down and write fewer notes. [. . .] it is now time to set an example of doing instead of documenting in part 3!

  Assessment: I tried and sometimes succeeded. Not a breakthrough or life-changer, but no relapse either. Call it good, 3/5.

- Remember at all times: Just keep killing your darling. A lot of delete is best for this project right now.

  Assessment: A lot got deleted, a little bit above expectation, but more could have been done, 4/5.

- Goals to measure: Keep implementation at 100%, test failures at 0, and start minimizing the size of zipped elements (the original driver of the whole hobby project). Meanwhile, it would be nice to bring down the number of unfinished parts, marked as fixmes and todos.

  Assessment: No problems - all according to plan (the 2 current test fails are deliberate and under control; Number of fixmes and todos grew, but this was "would be nice" in the first place). Call it a success 5/5.

Major steps I set in the end of August:

- Minimal bootstrapping code - excellent 5/5.
- Some optional Programming-Tools words implemented, but not all; Forth code refactored nicely but not perfectly - 4/5.
- Keep killing the darlings.. - I still suck a little bit in letting go of old stuff - 2/5.

- Prepare for more ultimate size minimisations, i.e., at least removal of unused words, automatic shortening of word names, inlining of constants, including hashed SDL&GL function names… grand success 5/5.

  I want to read Thinking Forth - not happen at all during this part, 0/5 !

  Get yet some more experience with Forth programming… - grand success 5/5.

How did I do with scheduling and general balancing of life?

- Surprisingly well! I was expecting a long black-out period of not touching the hobby project. But it didn't happen. It's been rather good since the beginning of September; all the balancing between work, family, and hobbies. Definitely better than expected. Not perfect, though, because of all the unfocused hours reported. Say 4/5.

So… what's the score?

All things considered - good or close to very good.

Where to go from here

The remaining list of undone and postponed things would be the initial short-list of what to do next:

- Read Thinking Forth.
- Dependency-graph solution for identifying unused words.
- Try making an even more compact "byte-string" approach to bootstrap and/or obfuscated code.
- Clean up the minifier a lot, or re-write totally.
- The load of small todo-notes goes without saying.
- The actual application programming.
- Maybe, at some point, revisit the note of 2020-08-19 about the order of NIP TUCK etc. in the tests… (I don't want to just moan without giving a suggested fix, so I'd need some time to think it through)

And now, the time is up. Full stop here for the 15 ECTS trilogy, and see you in Episode IV. I'll tag this version "POC-405h" because the project is now 3 x 135 = 405 hours in the making! Same number of hours that we expect as the personal input of a student taking our Software project course. Also, approximately the same number of hours we expect from our two first courses in programming.

Ok, I admit, I spent just **one** extra hour. Marking it in the next part, so I get an exact number here:

2020-11-22T12:15 (1 hour 135 total)

Part 4 / Episode IV: Thinking Forth etc.

(Learning diary of a hobby project.)

Goals

In this part, I pick up a strategy little bit different from that in the previous ones: I'm gonna start by reading Thinking Forth by Leo Brodie from cover to cover. After all, it was one of my original goals all those 406 hours ago. I'll do it one chapter at a time, discuss the learnings here, and do some exercises with my own codes here. It has become time to replace the placeholder proof-of-concept triangle visualization with something else - at least with a few more triangles. I'm quite sure 135 hours is enough to get to new places with both reading and some creative application programming. Meanwhile, I'll use a couple of hours now and then to catch up with earlier todo-notes and try to reduce the number of to-be-done items.

Copy-paste-modify-prioritize from the end of last episode:

- Read Thinking Forth.
- Actual application programming, having fun finally.
- Dependency-graph solution for identifying unused words.
- Reduction of the number of small todo-notes goes without saying.

Call it a success if the above gets done - next ones to be done only if time allows:

- Clean up the minifier a lot, or re-write totally.
- Try making an even more compact "byte-string" approach to bootstrap and/or obfuscated code.
- Maybe, at some point, revisit the note of 2020-08-19 about the order of NIP TUCK etc. in the tests… (I don't want to just moan without giving a suggested fix, so I'd need some time to think it through)

2020-11-22T12:15 (1 hour 1 total)

Picking up the book, and starting from front cover…

2020-11-22T17:00 (1 hour 2 total)

Leo Brodie (2004), Thinking Forth - A Language and Philosophy for Solving Problems. (Version 1.0).

I read the front matter and the prefaces for something like the 1000th time now. This time a couple of the names of the volunteer transcriptors of the 2004 open content version were familiar from the pages of the current Forth standard. I've pondered their comments and arguments, and copy-pasted reference implementations suggested by them into my own system. It almost feels like I know these people, even though we've never interacted and probably never will. Funny.

The textbook is now free and open, like all textbooks should be, in my opinion. In the case of Thinking Forth, the 2004 version seems to be the result of what looks like a loving community effort, too. I like the factory method of this edition.

The book content is now almost 40 years old, so it will be really interesting to read and contrast its arguments with what I've learned from other sources - I learned to read a couple of years after the first edition of Thinking Forth was published, and I've been learning and teaching programming using different sources until now. But this book is all new to me today.

Based on the original 1984 preface and the table of contents, I'm expecting to feel the "moisture of the waterfall design method" here and there. And some warnings against "picking up objects". But I'm happy that the book is posed largely as philosophy and opinion, which gives me a promise of a refreshing dialogue with grand wisdom from the days of the past. The greatest promise of being about human problem solving sounds like a topic forever interesting and timeless.

At the moment, I don't have time to continue - I'll have to start reviewing the many Master's Theses to be completed by the end of this year. But I'm very happy with this project right now: With the greatest certainty, I will soon have read Thinking Forth completely, as was the very original plan.

Next up:

- Read and discuss Chapter 1 before doing anything else.
- After reading, do some programming (like fix a couple of issues found earlier) for this own project; see if the doing becomes any different as a result of the reading..
- After some programming, go read the next chapter.
- Repeat until the whole book is done, one chapter at a time.
- Then, and not earlier, look at other goals and schedule of the project.

2020-11-23T08:48 (1 hour 3 total)

Chapter one: The Philosophy of Forth

Notes from the read

The chapter opens with a grand promise of examining some major problem solving philosophies and then describing a philosophy called "Forth". Does it live up to this? Have I already been exposed to this philosophy in practice over the 400 and more hours of "Just Tinkering Forth".

Notes to self… Brodie's breakdown of software elegance: memorability, power, abstraction (restriction vs. freedom?), manageability, modularity, writeability, top-down design (with costly patching), subroutines (ehm.. with fairly outdated reasons for costs), successive refinement (a parallel to top-down design?), structured design (simplicity, with 3 factors affecting modularity: functional [binding] strength, coupling, hierarchical input-process-output designing), information-hiding. End up with David L. Parnas' "uses" hierarchy and accommodating change.

Nice history with some outdated arguments. A canonical representation of history. Useful references from the mid-1970's. Beautiful and inspiring drawings. Should I have the courage to recommend this chapter as a reading on one of my courses? Perhaps for second or third year students, when they are familiar with the present day practices and able to view the text as historical to a certain extent.

I'll have a break from reading at this point, before entering the Forth part of the philosophy of Forth. Note to self: Approximately 2 hours spent with 18 pages of introductory, already familiar, material, careful word-by-word reading, no exercises. Just woken up, clear head.

This is pretty much a cliffhanger now: So far, it was the armchair history, only setting the stage for the next part which I expect to be the main points of the Forth philosophy… Interesting. But I'll have to read some Master's theses next, before coming back to the "Forth thesis".

Note to self: Were there still OCR errors here? "Contents or Register B"…

Reading pace: 2 hours / 18 pages (no exercises)

2020-11-24T08:10 (2 hours 5 total)

Ok, I rush back to this cliffhanger: How does Brodie project Forth against the software elegance of the first half of our engineering field? (writing this in 2020 and guessing most of Thinking Forth is based on the body of knowledge accumulated up to the late 1980's or so, and dawn of the digital equipment was in the 1940's; so it is roughly the first half of actual digital computing, yes).

I remember looking at the example ": BREAKFAST HURRIED? IF CEREAL ELSE EGGS THEN CLEAN ;" before seeing much Forth. I was very puzzled about the word order and troubled by the all-caps. But these might have been superficial thoughts evoked by an indoctrination to "void doBreakfast(){if (isHurry()) {cereal();} else {eggs();} clean();}" which, well, could puzzle somebody just as much with all the different parentheses and semicolons.

Out of interest, I'll go look at Appendix A to see how Brodie explains Forth to a reader who supposedly hasn't seen it before. I've had my own experiences of explaining "public class WelcomeToCS1{ public static void main(String[] args){system.out.println(”What the fffff“);}}" to people who have no clue prior to this kind of first exposure. Terrible things we have to do in our work as CS educators… I can relate to the "Python first" or, more generally, "scripting first" school of CS1 teaching. Drifting away from the topic of Forth, though. In reality, we are not likely to have many "Forth first" CS1 courses these days, so Forth will look odd to anyone who has been indoctrinated to pretty much anything else before.

Ok… Brodie's Appendix A is not for a total newcomer - it pretty much boils down to the final phrase, "read Starting Forth"… which I did earlier during this project. I think this point is exactly where I left off the first time around when looking at Thinking Forth. Back to the main matter… I think it would be pretty difficult to understand without "starting Forth" first, one way or another…

I can relate to Brodie's appreciation of Forth intrinsically addressing some of the elegance issues pictured in the initial, historical, framing. Also, the idea of "word" being a cool concept is quite convincingly laid out. Component programming and anticipation of change seems like an evergreen topic. The key philosophical thing here seems to be the "design by components" idea. A component whose 'how' is described by a lexicon is a stepping stone to Brodie's definition that "Forth is a programming environment for creating application-oriented languages. (That last sentence may be the most succinct description of Forth that you'll find.)"

I can see that. And I love a succinct description. Now I'd like to see where that leads up to. Brodie says "We need only enjoy the ease of describing an application in natural terms". The example of RED and GREEN APPLES is quite limited, though, in my opinion. The real beauty must lie further down the road… and the longer road must be difficult or perhaps impossible to fit in a succinct introduction. So I'll just SWALLOW the any-colored APPLE for now and move on.

I think many of the properties of Forth highlighted by Brodie, such as its interactive nature, are not very relevant selling-points nowadays when other languages have caught up with interactive REPLs and sandboxes. The type safety issues remain. So, the arguments of enduring value, in my opinion, are those related to flexibility, extensibility, and the lexicon idea.

It is easy to see why people could have likened Brodie's ideas with extreme programming or agile methods. In a way, it seems that other programming platforms and design practices have become quite Forth-like over the years, trying to reach towards the ideals of agility and interactivity, among others.

So… what is my take home message from Chapter 1? Not much yet, I think. Somehow my thinking or perspectives did not change dramatically, not about Forth nor programming at large. I think I got a glimpse of the philosophy behind Forth, but I think I had learned that already from the past 400 hours of implementing Forth and the readings from Jonesforth source and Brodie's Starting Forth. I am yet to be mind-blown.

Right now, between Chapters 1 and 2, I think it is time to have some fun with actual Forth. Maybe I'll develop some insight into the text while looking at my own "lexicon" under development.

Reading pace: 2 hours / 17 pages (no exercises)

2020-11-24T17:27 (2 hours 7 total)

I think I went a long way towards building a dependency graph that can be used to exactly determine the unused words that the obfuscator can omit. Just 2 hours. But let us note that calendar time was needed for the idea to build up in the head. I don't think I could have done a thought so deep during one working day, for example. Things cook up in the head over time. Give yourself the time to think! But also tinker with the ideas and test them in practice! Hmm… I think I'm actually pondering the design ideas from Thinking Forth right now, somehow subconciously. Because I just catched myself writing about time used in thinking vs. doing (analoguous to designing vs. implementing).

Well.. Anyway, now I go to work for today and perhaps the rest of the week. Long sessions ahead on the Master degree orientation course. Note to self, when I get back to this project:

- Just finish with an initial version of the whole dependency graph thing. (Pretty much done by 2020-11-29, in approximately 9 hours of approximate "time on this project")
- After doing that one thing, but nothing more than that thing, go back to Thinking Forth and process Chapter 2.

2020-11-25T08:51 (2 hours 9 total)

Wow, it was a job of less than 20 minutes to hook the asm hack dependency dumper to the Forth-side interpreter.

And another wow, I've got quite clean and useful-looking output before even one hour got full.

I will have some trouble with creating words, starting with the "CREATE name n CELLS ALLOT" idom itself, and VARIABLE and then any arbitrary words that do the CREATE DOES> business… I think that I will cut all the possible corners here, and just have a list of idioms (regEx?) and a list of creating words as required parameters of the obfuscator. User can do that much of manual work? I'll see about that later when I'll be that user :). Fully automatic methods would need a lot of more thought, I think. I'll be happy with a hack that more or less works, at least in the beginning.

I need to pick up these thoughts after some nice work-time again, though.

But this latest morning hour felt super efficient again. I'm happy about being able to do even an hour now and then, but on a seemingly regular basis nowadays.

2020-11-25T08:12 (1 hour 10 total)

Language revision of above notes. Reminder to self: More doing, less writing!! Two hours spent with not progressing with the POC.

Focus!! Continuing with the dependency graph thing. Mostly thinking, but also some code.

2020-11-28T12:45 (3 hours 13 total)

Some minutes here and there - maybe an hour's worth, spread over the afternoon. Furthered the obfuscator. Beep and triangle getting much smaller than before, almost automatically, but this is just the beginning:

4032 packtest.obfpoc.gz

Much more to be done.. but how much do I want to press onwards before just reading more again?

2020-11-29T00:09 (2 hours 15 total)

Silence all around, some coffee, and just a couple of beers. At "B. peak" of xkcd (https://xkcd.com/323/) or at least that was the plan. I hope it stays this way for a while.

Nice and quiet, yes. Getting things done.

Run the numbers as of now:

2510 packtest.felf.gz 4876 packtest.core.gz 7064 packtest.poc.gz 3788 packtest.obfpoc.gz

The proof-of-concept beep and triangle now fits well under 4k, and it has a lot of stuff that can be removed and optimized in so many ways. It is quite a big bulk still, but having it under 4k is a promising sign. At least I can now be certain that some kind of an entry will fit in 4k for sure. I can rest assured the next time I rest…

Tried some more quick removals of stuff, and got it smaller again:

3737 packtest.obfpoc.gz

Now I don't have any names longer than 2 chars after obfuscation. The end result seems to be quite neat, lean, and especially mean.

It would be nice to just play with the toy once again, but now I must command myself and observe that I have done the one thing I was supposed to do before immediately returning to the task of reading Thinking Forth!

I'll just make a note that when I come back to programming the next time, after reading Chapter 2, this is where I'm at:

- Upgrade the proof-of-concept to something more aesthetic. It can still be a beep and a triangle, but perhaps just a bit more eye-pleasing than it is at the moment. Could this be the beginning of the much anticipated "Actual application programming, having fun finally"?
- Start shooting down every piece of code that is not absolutely necessary for the audiovisual tiny exe. Should start getting smaller packed exe without

too much manual work already - I hope… if not, then I'll find out some more necessary steps.

- And, of course, if possible, shoot down some of the earlier FIXMEs and TODOs already? Let's see what is possible..

2020-11-29T03:00 (3 hours 18 total)

Chapter two: Analysis

Notes from the read

From analysis to engineering to usage, but on many levels at a time. Again, I find myself in that part of the audience who thinks about agility, fast prototyping, and test-driven development, and the like, when I read about the multilevel application of the development cycle. But, indeed, the difference between Forth and other tools may be found in the necessity and the intrinsicality of the short design cycle in creating each word - "on the order of minutes" says Brodie.

"The iterative approach" that Brodie cites from Kim Harris, 1981, looks much like the method I've come to know as design science. From the analyze-design-implement-test cycle one could even see an abstracted cycle of "observe X", "think about dealing with X", "try new things regarding X", "evaluate and analyse X again" which could be as much about learning or growing up as a human being, organism, organisation, or other learning system, as it is about engineering. From the philosophical point of view - is there any difference at all, or should we say that engineering is learning. What about the other way around: What would it mean to say learning is engineering? Let me check the etymology of the word engineering - oh, "constructor of military engines of war" … not a very happy notion… sticking to the present-day use of the word, I'd say that there is at least a lot in common between human learning in general and the work of an engineer towards learning about the requirements, design, and implementation of a technical device or software. But we should make machines of peace, not war.

Brodie's comment about evolution is far-fetched. Certainly, a Creator, if we think of one, is not a top-down designer. But we must observe the practically infinite inefficiency of the method of bottom-up creation via evolution: It took billions of years to develop multicell life. As far as we know, it may have taken the full capacity of a complete Universe to create sentient life on a single spot. The waste of resources simply could not be any larger. Also, the only design of life we know of may be flawed from the start, because it may end up destroying itself via the catastrophic process of becoming industrial and deleting its habitable climate in a billionth of the time it took to evolve it. So, let's not compare evolution and software development. The testimonials of software developers are much better suited to matters regarding their work than comparisons to the origins and destinations of life.

Oh, there are boxed tips in the book. That's very nice. The first one is "Start simple. Get it running. Learn what you're trying to do. Add complexity gradually, as needed to fit the requirements and constraints. Don't be afraid to restart from scratch."

This is a nice thought that I, too, recite to my students in every possible occasion. I call it the "step one" of the "canon of problem solving". It is nice to see that after the digression to matters of creation and evolution, we're once again discussing the human task of solving problems - a task that canonically requires understanding the problem, which in turn requires learning and managing complexity by decomposition. And all that requires patience from the person doing it. So, "Start simple. Get it running. . . " is a good tip. But this is canonical and can be done with any programming language. Let's see where the story about Forth goes from here.

I think that the interviews of software developers are of much value here. Real testimonials to the canonical procedures. Planning is of value. Understanding the problem. None of this is news in 2020, of course.

Btw, my guess is that this John Teleska has been doing more psychology than software since the 1990's (http://www.unconsciousresources.com/aboutJT.html). That's fine; his words about being uncertain about uncertainties, giving time for specifications, and doing feasibility studies seem to be in line with the enduring canon, too.

The present-day Internet doesn't easily give information about Michael Starling. Peter Kogge has been quite influential in processor design at IBM. Dave Johnson might be the David A. Johnson, co-author of some scientific papers published during a few decades. I had to check these people, of course, because we have Internet now.

Some nice quotes.. "you don't know completely what you're doing till you've done it once" (Michael Starling). "the best way to write an application is to write it twice. Throw away the first version and chalk it up to experience." (Leo Brodie) "When I build such a prototype, I do so with the firm constraint that I will use not a single line of code from the prototype in the final program." (Peter Kogge)

But once again, this seems to be embedded deeply in the agile practices of today.

Analysis goes from requirements and constraints to a conceptual model (interfaces) and then a design. Dynamicity is important. These themes have endured time since the 1980's, I believe.

Tips 2.2–2.4: "Prototype". Then Tip 2.5 is "Strive to build a solid conceptual model before beginning the design." because requirements and constraints are easier to fit into a model than into a design. Makes sense. Again, I think these ideas have since been engraved in our minds in our education, under the title of agile.

Error handling is a key difference between coding for oneself and coding for another. Structured analysis and data-flow diagrams (DFD) etc. important for communication with the customer. DFD's sort of implicit in Forth programming because "Forth, used properly, approaches a functional language."

Yet another cool quotation (by Brodie himself): "Visualization of ideas helps in understanding problems, particularly those problems that are too complex to perceive in a linear way." This is one more of the things I consider canonical and try to have my students learn at every possible situation.

Tip 2.10 is a message, posed by Brodie in an argument against some design literature of the time (1980's): "You don't understand a problem until you can simplify it." There is a good point here! Perhaps something to recite to the students of today.

The running example of land-line phone costs is a fresh blast from the past in these days of 5G and unlimited data subscriptions. But the topic of not-so-easily described operations is a lasting one, for sure.

Aaaand.. there it is, as Tip 2.11: "Keep it simple." The one rule to rule all the rules, in my opinion. Of course, it needs to come with further salt like Brodie's Tip 2.12 about changeability: "Generality usually involves complexity. Don't generalize your solution any more than will be required; instead, keep it changeable." More salt to the mix: demand explanations from the customer, find out "don't cares", … Be tactful and non-threatening with a customer, take advantage of what's available…

A nice quote from Moore: "It's much more fun redesigning the world than implementing it."

2020-11-29T22:00 (2 hours 20 total)

A little bit slower pace per page - but I did read and make notes, and also I did the additional "exercise" of searching the web for the people interviewed by Brodie in the 1980's.

Reading pace: Pages 38-65 (27 pages) done in 4.5 hours

Now, off to work. Come back to page 66 later.

2020-11-30T11:30 (2.5 hours 22.5 total)

Tip 2.18: "Everything takes longer than you think, including thinking." I think I've independently discovered this reality, and I wonder if somebody has ever been able to learn it from a book..

And a recommandation to read The Mythical Man-Month by Brooks. It's been on my to-read list for some decades; Thinking Forth won the race of being the book at hand.

Then I did check out LaFORTH (http://www.forth.org/Ting/LaForth/) mentioned by Brodie. I think it is a great testimonial to the free spirit of Forth - if you don't like the standard, just do it your own way.

I’m now done with Chapter 2/8 (roughly 25%) of reading in 23/135 hours (15%) of time allocated for this “course”.

Reading pace: Pages 38-72 (34 pages; some blank in range) done in 5.0 hours

I’ll go to programming / self-decided practical labs for just a couple of hours. Let’s pick one particular task, once again, and remember Brodie’s Tip 2.17 - a 2-hour addition takes 12. So think small here.

Ok, I’ll pick one small thing from each of the tracks I want to push forwards, as identified on 2020-11-29:

- Some kind of change to what the proof-of-concept looks like, and make it independent of aspect ratio.
- Shoot down 100 packed-bytes worth of unnecessary code, so the new and improved poc is smaller than the current one. (current 3737, target below 3637)
- Handle 5 FIXMEs in code. (currently 59 FIXMEs, target 54)

Try to reach these goals but do no more before coming back to read Chapter 3.

2020-11-30T21:45 (0.5 hours 23 total)

Intermediate Surprise Exercise: Think about what I learned from Chapter 2?

I think this question should always be asked. I almost forgot.

In sum, I found not much in Chapter 2 of Thinking Forth that hasn’t become part of the software engineering canon that I have been taught as fact on the 21st century university courses. Now I teach the same canon. The interviews of real system developers of the 1980’s and the short, funny, and insightful tips Brodie has derived from them seem to have lasting value as exciting history of the real world of problem solving for customers.

The comparison between “the traditional way” and “the Forth way” has lost its edge over the decades, because much of the Forth way seems to have become mainstream and new tradition since the writing of Thinking Forth. This observation highlights the original brilliance of Forth and the book, of course, but today I think we need to read it in a historical context.

What can I take home, personally? Trying to never say never, I think I have a very small probability of ever working as a system designer or lead developer responsible for the requirements, design, and delivery schedule of a customer project. Yet, there is a quite high risk that I need to teach these topics “in theory” and simulate practice on university courses. If not software, we do design and deliver things of value all the time: courses, research papers, R&D artefacts. So, any new learnings and insight is necessary. The “tips” of Brodie are a nice reminder of the things we should always keep in mind. As said, nothing new, but some things just can’t be repeated and recapitulated too often.

As for software development, I do only the present kind of solo projects in which I don't have to worry about a customer or end-user… or a group of other developers for that matter. So I'll be pretty humble about the learnings that remain mostly theoretical before the need to do things in real life suddenly emerges. I better not say it's never.

But I do have a design in each solo project. In these demoscene competition things, I have many requirements and constraints, some more extreme than others: Trying to make interesting content that fits in restricted-size executables. Creating code that can be re-used in a next year's competition or by some other developer… or by audience in a workshop or on a course. Ok, maybe I shouldn't be too modest about my goals or their imagined disconnection from other people. Even these notes I apparently write for myself but I also want them to work as examples to others who wish to pursue focused self-study efforts…

How do I fare, personally, in light of what I just read:

- My requirements and constraints in this project: Externally defined by demoscene compo rules.
- Conceptual model / interfaces: Striving to be standard in all directions
- Simplicity: Always a goal, always trained further, always difficult to achieve.
- Budgeting and Scheduling: "Good, Fast, Cheap - pick any two" (from the Michael Starling interview in Chapter 2). For this project, I picked Good and Cheap. I never planned to be fast. I did plan to have a milestone publication within a year, for Assembly 2020, but then there was COVID-19, so I think what I did was just a very successful re-budgeting and re-scheduling according to uncontrollable events.
- Documentation: You're reading the soft part of it - what happened in the brain. I think I manage, but the final decision is up to the reader. The hard documentation is in code comments. I think I manage, too. I don't aim to be as educational as Jonesforth, but I think I have left enough clues as to the why's of everything, …

I don't know if this recap and half-finished thinking appeared to be useful. But I tried to at least remember to do the reflective part of learning. I maintain that it is important. My own Tip:

When learning, occasionally stop to reflect and recap - do not just rush forward.

It comes with a parallel:

When delivering, you are not learning! Don't think - just ship! Learning can only truly happen when you don't have the pressure of having to deliver by a deadline.

This difference between learning (studying) and delivering (working) should be kept in mind, at all times. Maintain the difference. Corollary:

When studying, don't work. When working, don't expect to learn (about things other than "working" itself, of course; to learn how to work is one important thing, too, but you can leave all that stuff to the working place).

Ok... I did get to a crystallized thought in the end. Whether in relation to what was read, or not, but my thinking made my brain move an inch. Nice. I'm learning now, and definitely not working.

That was studying, with reflection! Now I go to work, to deliver!

Note to self, when coming back here: Deliver for this project, too :), refrain from reading and reflection for just a bit. Step forward in the iterative process of practical learning through experience. Have fun, too.

Exercises

2020-12-01T09:21 (2 hours 25 total)

Reading the proof-of-concept code. It has been a while since the last time, so I'll have to re-familiarize. Fortunately, I've left comments. But it will take time. Starting with very small re-touches and thinking about structural improvements. I need to make small steps through the mess. So, I need to figure out what the steps could be.

Haha.. I'm going the easy route - some of my FIXMEs are much related to the size, so I'll get these two improved at the cost of one at the moment. The 5 fixes got done already while also getting the size improvement:

3621 packtest.obfpoc.gz

Next up:

- Some small aesthetic improvement of poc within approximately the current size. (Still thinking aspect ratio handling here). Then back to read Chapter 3.

Side note: How much in 1 ECTS credit this time? Reading 60 pages of easy text and some reflection. Not much! Must remember that 1 credit is for really small stuff. And vice versa: Big stuff needs bigger reward. And then there is the matter of perfecting the language and stuff. I must learn to just let it go sometimes.

2020-12-02T08:52 (2.5 hours 27.5 total)

Done. It is still a beep and a triangle, but the triangle is now centered, keeping shape, and rotating around, and not disappearing to black every now and then. Reduced fix-needs to 50 and size to 3629.

Time spent with code, between book reads: 5 hours; quite tolerable.

Now, back to reading. Next, Chapter 3. After reading, come back to code towards the following:

purpose.

- Try to take the readings into practice, somehow.

2020-12-05T22:58 (2.5 hours 30 total)

Chapter three: Preliminary Design/Decomposition

Notes from the read

The opening, once again, is a case for iterative design. “a component need only suffice for the current iteration’s design” says Brodie. An example of a text field editor is given - decomposition by component gives a few tiers of lexicons for data, operations, and user interface. The key point is to prepare for reuse or especially change. Once again, the design principles seem very familiar from what we have been taught in the 21st century. Let’s see what the comparison to tradition (as of 1980) adds to the picture.

Some highlights of the text: “. . . control flow structure, a superficial element of program design . . . By decomposing our application into components which use one another, we achieved not only elegance but a more direct path to correctness.”

Falling asleep while trying to read about the Interface Component.

2020-12-06T01:40 (2.5 hours 32.5 total)

Re-reading some wide awake, with a clearer head and, unsurprisingly, faster understanding.

Tip 3.4: “Both data structures and the commands involved in the communication of data between modules should be localized in an interface component.”

It all seems to be about localization, information hiding, and asking questions about relations of items while designing the whole. Brodie’s example of his hobby project yields the principle: “The moral of this story is that we must distinguish between data structures that are validly used only within a single component and those that may be shared [in objective units!] by more than one component.” This moral has been deeply etched into our minds in the context of object oriented design, years after Brodie’s text and critiques of the OOP ideas.

Decomposition by components was one way, and another is decomposition by sequential complexity: The basic flow of a program source structure would be from elementary definitions to more and more complex ones, which is natural in Forth that builds definitions based on earlier ones. But some later-defined actions should be invoked by earlier-defined ones. Natural ordering by sequential complexity can be preserved by using “vectoring” which I’m guessing is what we’ve been taught under such names as “hook”, “call-back”, or “event handler”. Brodie will come back to the topic in a later chapter, so we’ll see if it is the same stuff.

Now, I’m off to handle some real-life interruptions; coming back shortly.

2020-12-06T15:22 (1.5 hours 34 total)

In the subsection about limits of level thinking, Brodie asserts that some conceptions of low vs. high level software (presumably as of early 1980's) are misconceptions that distort thinking.

I really love the interview bits, and now there are some more with Moore. A nice quote (from Moore) is "I know where I'm going so I don't have to start there." In this current project of mine, I totally relate to this quote. When there are no questions about the goal, it really makes no sense to start iterating top-down, does it? And Moore continues with another very important point. "I'll do the thing that's most fun in order to get into the problem. If I have to clean up all those details later, that's the price I pay." Then he continues after Brodie asks if "fun-down" is perferrable, "If we were giving a demonstration to a customer in two days [. . .] I would start with the most visible thing, not the most fun thing." So, I read his message as that of getting things prioritized and done in an "it all depends" fashion. Brodie's multi-bulleted Tip 3.6 pretty much boils down to "it depends", or "do what is necessary in your specific case".

I'm gonna copy one more quote to these notes. This one is from Brodie: "Thinking in terms of lexicons is like painting a huge mural that spans several canvases. You work on all the canvases at once, first sketching in the key design elements, then adding splashes of color here and there. . . until the entire wall is complete." This is a nice philosophy, and a nice wording for it.

Then there is the rant about dangers of objects, with a negotiating note by an editor (Bernd Paysan); I guess we have to filter these original thoughts, and interpret them from the current point of view and in light of what the world has learned about objects later on. Forth or not, the utopian vision of ". . . application programmer could either rewrite the driver or write an extension to the driver. . . " is wonderful. The whole free and open source movement has done a lot, and giant closed-source companies have made sure that. . . the dream remains a dream, and we still have reasons to wake up in the morning to go wave our flags on top of barricades in the 2020's. One day, the application programmer can rewrite the driver or write an extension, "using available tools from [the hardware manufacturer]".

Done with Chapter Three. . . All in all, I read this one mostly as a rebel statement, a critical voice, and a free-minded yearning for open designs and customizable environments. I think the world has caught up with many of the reasons of critisim since the 1980s. The Forth thinking, as described here, has become mainstream, even if Forth as a language has not. But surely, it has been a longer and more tedious route than would have been necessary. Perhaps more corporate executives should have read Brodie in the 20th century?

2020-12-06T23:40 (1 hour 35 total)

Oh, exercises with answers in an appendix. Nice surprise! I'll do some, just for fun. Naturally, I won't spoil the exercises by any answers here.

Wow, the second question is almost two pages, with example code, and requires reading and understanding much of the story from dozens of pages. I think many of my current students would get frustrated in the middle of the question text and complain about its length. Attention span has been reduced since the 1980s?

The questions and answers did support the main ideas of the chapter. It would have been nice to have more exercises - I totally understand why my students ask the same of me all the time, and I haven't had the time to figure out more exercises than there is… so, Brodie is totally excused by me to not have more "further thinking" exercises here. It takes massive amounts of time to create useful exercises.

Reading pace: Pages 72-98 (26 pages; some blank; some exercises) done in 5 hours. Getting slower? There were some tired and unfocused attempts here. Yet I need to remind myself that it is not so easy to find the focused and alert time. So… I'm sampling, and I need to understand that I have lived twice as long as the youngest students I have today, so I must have at least twice the experience in getting focused when need be. So… I'll need to tune down some of my own course requirements, perhaps. We'll see. I'll have to think about this. Now, I'll just collect some data on my own performance.

Take-home from Chapter 3

Technically, within the realm of Forth, Chapter Three was about best practices of doing preliminary design via decomposition. What do I take home to my current project?

Not much as of yet, I think. I'm doing a hobby project here, and I know my goal. So I'm going around it in a hobbyist fashion in the "fun-down" approach which really isn't about getting the design right or showing progress overnight. Also, there's a lot of learning and necessary side-tracks involved. It would be really awkward and unfitting to simulate a design for any other purpose than I actually have. I think that was part of the whole moral of the chapter by Brodie, too. I'm doing my business, and I'm doing it in the way that supports attaining the goal. In my case, it is "fun-up", "learning-up", "bottom-up", "just get something running", and so on.

For the computer science student of 2020, I don't think there is much news other than what I have recapitulated here for myself.

Ok, well I did some more thinking, somewhat forcibly, actually leading to a pro tip of my own, to the student and to myself and to everyone:

Do stop to think every now and then. (You may have to force yourself, but it is usually worth the trouble in the long run!)

So… what was the thought right now?

I do have a clear anticipation of change in my project. It has been implicit, but I need to make it explicit to not end up in trouble at a later point, I think. I want to be independent of the platform - I'm developing on Linux, and I don't have much interest in making pretty much anything under other operating systems at the moment. Yet, I do want my code to be portable. Of special interest is the UI and graphics. I'm planning to use SDL2 and OpenGL on Linux, but really I want to avoid my application code being too dependent on the choice. Ok, OpenGL will probably be unavoidable due to GLSL shaders that are probably going to run most of the graphics. But other than that, I should try to separate the "what" from the "how". As another result of the forcible thinking, I'll draft a top-down preliminary design here by identifying some components or "lexicons" in Brodie's Forth-parlance that I'm likely to need:

- Window and UI lexicon (something parallel to the SDL2 interface of a required subset, but really not necessarily tied to particular names of symbols used in the SDL2). In Forth, I should define descriptive names that hide the platform of choice.
- Audio generation lexicon, divided to synthesis and music lexicons (I've been dreaming of a "language" similar to the pattern layouts in "tracker" software I grew up with as a youngster)
- Sync tracker lexicon (The dream of tracker layout continues with the idea of using words parallel to GNU Rocket that is being used for demos by many sceners these days)
- Graphics output lexicon (a triangle or a mesh thereof, or a back to front buffer swap, or adding overlays, should not depend on names dependent on a device or a library)
- Audio playback lexicon (In my current dream, I'll be happy with start/stop/wind-to, but still, this, too should be independent of platform)

Yep, there it is now, implicit goal setting explicitly written out as some sort of a top-level "design".

Do some code

All that read and said, let me have some fun with the code again. Next up:

- Rid of 5 more FIXME-notes (target: 54-5 = 49); start with an alternative for null-terminated strings
- Make it smaller (target: 3500 bytes); likely to correlate with doing the above-mentioned fixes, at the moment.
- Maybe try to think in terms of "lexicons", "components" and "interfaces" as in the just-read chapter by Brodie. Does it imply some changes to my current proof-of-concept / platform code? Where is my likely change or reuse? How to best accommodate those, with my current understanding.

Don't spend too much time trying as of yet… the story continues for more than half a book still.

- Quite soon after just achieving the above items and not more, go to Chapter Four to do more reading!

Ok, I have to admit that there were 5 more or less trivial or already fixed points. So I'm down to 49 remaining FIXME-notes in less than 30 minutes. It almost feels like cheating, but I did agree on my own metrics and goals, so pushing forward to size improvement. Nah, I also did the third bullet point (which was some thinking), and appended the resulting thoughts to the note above. Only the size improvement

That's gonna have to wait until tomorrow, though. Need to sleep now, I guess.

2020-12-07T01:39 (2 hours 37 total)

Didn't wait, couldn't. Tired and suboptimal time. Can't make any big decisions or think long thoughts, and even debugging in the small takes too much effort. But I have the quiet world around me, so I'll

nice new size for the beep triangle:

3285 packtest.obfpoc.gz

I went through the remaining FIXME-notes, and they are not as easy as the ones so far. I need to be very careful in what I require of myself between the next coding parts between chapters. Can't be 5 FIXMEs at a time anymore - that would likely take too much time. I'll do the fixing later (or never..)

And now: to sleep, really, really, to sleep.

2020-12-07T03:59 (2 hours 39 total)

Chapter four: Detailed design / problem solving

Onwards, to read pages 99-134. It will be pretty much midpoint of the book, after this chapter, based on page count.

Notes from the read

Chapter four opens with general problem solving methods. Let's see if Brodie's exposition is in line with what I've been teaching to my students as the Basic Canon of Problem Solving. Yes, it pretty much is. I like this quote: "Use words, phrases, figures and tables, or any kind of graphic representation of the data and/or rules to help you see the maximum information at a glance. Fill your mind to bursting with the requirements of the problem you need to solve, the way you might fill your lungs with air." The story continues from there, of course - according to the canon.

Tips that describe possible moves in detailed problem solving: "Analogous problems", "work forward", "work backward", "believe", "recognize the auxiliary problem", "step back from the problem", "whole-brain thinking", "evaluate", "look for alternatives"

Brodie cites a 1975 book by Leslie Hart, "How the brain works". Might be an interesting read - but, well, there are so many books already on the to-read list; in 2020, I'd be ashamed not to pick up Kahneman's Thinking, Fast and Slow as the first next read about how the brain works. I'm quite ashamed to read so little, but that is another story… I'm working on that right now and right here with Thinking Forth!

Another nice quote from Brodie: "Don't invest too much effort in your first solution without asking yourself for a second opinion."

Wow, next, an interview. I love these.. Very nice testimonial from Donald A. Burgess, about drawing little men, and leading up to "Your head is a whole lot better than the computer for inventing things". (I did have to check: the present-day Internet seems to be very silent about a Donald A. Burgess or Scientek Instrumentation Inc., but I have no doubt that he has been a good choice to be interviewed by Brodie in the 1980s.)

Then, there seems to be a more practical section ahead of me, titled "Detailed design"; I'll have to come back to it after work.

2020-12-08T08:40 (1.5 hours 40.5 total)

Brodie's tip to rule other tips in detailed design using Forth: "Each definition should perform a simple, well-defined task."

Then there are eight steps, starting from external definitions and working backwards from there. The tip for external definitions is "… create a lexicon that will make your later code readable and easy to maintain." - somehow that sounds like one of those things that are extremely much easier said than done. I think it is a universal fact of life in design.

The Forth word order is funny, as Brodie paints in his Figure of "lawn mow fence paint post fix". I think it is not a very big deal for a programmer, but I can understand the discussion of syntactic difficulty from the point of view of a customer or end user of a domain language. To keep the learning curve gentle, domain language should be close to the native language of the end user. A tiny problem then would be internationalization and coping with the differences of grammar. While a prefix language expresses "in the Foo", we in Finland talk in postfix "Foossa", and so on. Just maybe.. a domain language, when not customized to each language in the world, should be like an "Esperanto of commanding the machine" - not really in the likeness of anyone's mother tongue? At least until we get the chatbot AI to really understand natural language :). From what I see in 2020, we are not there yet. I'll be happy with designing my demoscene intro lexicon using the Forth-friendly grammar of "number name operation" and so on.

Brodie's tips for Forth continue.. At some point it would be nice to just gather all the tips into a checklist of Forth style. Aha - indeed, the section on Detailed design is like a list of good Forth style conventions. Most of this must be rather nonsensical to a reader who hasn't done some Forth before, possibly reading Brodie's Starting Forth before entering Thinking Forth… Just a word of caution to the reader of this current text - you can't expect to just jump into advanced level before doing the groundwork. There, I just recited the canonical word of caution to anyone trying to achieve anything :). Good. Go implement in practice and do the work - from ground up..

Nice quote from Moore again, yet too long to reproduce here; refer back to "A simple solution is one that…" in relation to solving a problem vs. writing an interpreter.

The sections about algorithms and data structures and preferring calculation over data structures over logic brought not much news. The emphasis on *adequate* in "the best solution to a problem is the simplest adequate one" is something to live by every day.

(Come back to page 122, practical example, after work and other real-life interrupts.)

2020-12-09T08:30 (1.5 hours 42 total)

The example of printing roman numerals is mostly an example about problem solving - well, it concludes a whole chapter dealing with problem solving. In the context of Forth, it boils down to figuring out a lot of simple, irreducible, internal words behind the lexicon, and using stateful data structures. The thinking, goal, and means seem to be pretty much the same that we teach current students today. So, in 40 years, what has changed? In the way that people use their brain to solve problems, not much, as expected. The programming languages and environments, including Forth itself, have changed a lot, which makes it a little bit hard to relate to the example that has a sort of Back to the Future klang now. This book is not for the newbie of today.

The "right-brained" thing was all new to me - seems like some 1980s myth more or less busted later on. But the tips about visualizing problems as drawings of mechanical devices seems like a useful tool, no matter which side of the brain would dominate the process. This is what I try to tell my students, too. If drawing a three-handed arm helps in figuring out a solution, it was a very good thing to draw.

The parallels drawn between programming and the scientific method are something to ponder at some point in the long run.

Again, there is just one exercise, and the shuffle algorithm is a bit old to be of interest when we're not in our CS1 anymore. It is a tiny bit funny how Thinking Forth is sort of written for novices about problem solving and algorithmic thinking while it sort of requires some pre-requisite knowledge on how computers and programming work in general. But in 1983, this may have been the coolest thing around. I don't know, I was 4, and later on the programming books in the public library were all about BASIC and the junior computing courses about Pascal… Somehow we never really heard about Forth even though I can now totally understand the claims of its superiority to other alternatives of its time, dubbed "traditional languages" in Thinking Forth.

Yes, the shuffling algorithm.. I think I'll save some time here, and skip the coding. The thing is quite visual in my head from earlier learnings - also the way it would look like in Forth… Check Appendix to see if I wasn't hallucinating. I wasn't.. but..

BEWARE: The example answer in Thinking Forth is a dangerously flawed algorithm that doesn't do an unbiased shuffle!! Explanation of what is wrong with it: https://blog.codinghorror.com/the-danger-of-naivete/

It is best to learn algorithms from algorithms textbooks and safe designs from textbooks on safe designs. And use tried, true, and proven-correct stuff when those are available. Let us pardon Brodie for the rigged deck of poker cards, and trust that they haven't used this particular textbook example in your favorite Internet casino!

Reading pace: Pages 99-134 (35 pages; some blank; practically no exercises) 6 hours. Interestingly, a quite stable pace. I don't think I've ever measured my reading this way. Might be interesting.

I'm halfway through the book and about a third into "course" hours. So, it is going well. And it's not in my mother tongue. And I'm getting old.. can I ask my students to keep up with similar pace with Finnish course material? For sure! With exercises / coding projects? For sure! Quite reassuring this exercise is, professionally.

Today was nice for this project - many hours of quiet time, even after office hours. Not very common, so I'll celebrate the occasion now.

Hmm… time to decide whether to go to sleep or try to go on. Getting tired. Next it would be back to programming. Perhaps it is better to sleep and dream about the new learnings. Relax, and let the subconscious mind do its work.

2020-12-10T00:30 (3 hours 45 total)

I continued, because the flow was stronger than the wise understanding of the need to sleep.

Skimmed through the assembler implementation. Quite many FIXMEs are actually questions of doing things this or that way… easy to do technically but difficult to decide. Can't do when tired. Must wait for some of the morning-hobby-hours to appear. They have been delightfully common as of late.

2020-12-10T01:54 (1 hour 46 total)

Take-home from Chapter 4

This was a chapter about problem solving and creativity. Forth was a vessel, and a by-product, yet a central character. Like a rabbit?

Coincidentally, earlier this week at work, I came across the debated doctoral self portrait of professor Riitta Nelimarkka. What a wonderful read, although I only spent time barely enough to sample it here and there as of yet. Every page is just full of quote-worthy insight on art and creativity by Elise and the Rabbit.

At this point, half-way through reading a philosophical programming book from 1980s and writing down this travelogue, upon a chapter on problem solving, I totally resonate with the messy friction of Elise, Rabbit, Ball, and Fluegel while being in this rather lonely place on my self-chosen journey towards an unknown, perhaps non-existing, reward. Before Rabbit runs into the door, we hear, on page 31: "If only some Australian rabbit would say even one encouraging word now. It seems unreasonably lonely to continue this expedition floating in space towards fading oneself out, and that has been said only so that the project would not seem narcissistic. Big Cosmos! I did not remember that the sky here isn't blue but black."

I resonate.

The Day of Finding Purpose

Mark the date - I had a long and deep, yet nascent and formless thought, in which I found the underlying purpose of this self-indulgent trip of mine. It is the same, I guess, as before, but never clearly stated in words. The thought is to develop in and around the neocortex, in whole-brain, not right nor left alone, to be expressed in the final conclusion of this project, some 200 project-hours in the future from here.

Noting to myself some keywords, for returning to the thought: relation of art, programming, science, engineering, and problem solving; values in higher education. Purpose-first approach (Kathryn Cunningham), philosophy of Leo Brodie, Riitta Nelimarkka, Timo Laine, and contemporary learning theorists. And that of my own, in development, as I code along for demoscene in a language of marginal interest!

Now, off to work for today. With a sense of purpose.

2020-12-10T08:49 (2 hours 48 total)

Code some more

Let's go easy:

- Just fix something, at least one thing.
- Make it smaller, any number of bytes smaller

- Try to think “lexicon” and “components” and “principled design” in arranging the proof-of-concept code.
- Possibly don’t use so many hours before coming back to reading; after all, the interesting and probably most useful part of Thinking Forth is yet to come.

Situation at start:

3285 packtest.obfpoc.gz nFIXMEs: 49 #TODOs: 65

First morning hour got me here: 3201 packtest.obfpoc.gz nFIXMEs: 48 #TODOs: 62

Too easy. Must try to do a second one :).

2020-12-11T08:16 (1 hour 49 total)

Carrying on.. still finding some easy ones amid the hard ones..

3197 packtest.obfpoc.gz nFIXMEs: 47 #TODOs: 58

2020-12-11T22:20 (1 hour 50 total)

Fixed 2, then marked 1 more to be fixed. That’s how it goes.

3146 packtest.obfpoc.gz nFIXMEs: 48 #TODOs: 58

I want to get the assembler side finished next, but it will need coding on the Forth side at the same time to maintain functionality. It will be a hard cognitive task that will take a couple of hours. Assess my current condition: I’m tired, not capable of hard cognitive tasks. So I decide to defer the coding to future hours with a clear head. I can read and acquire knowledge but not solve puzzles. So, I’ll save the project hours for the next coding phase. Then, I’ll set the goal as follows:

- Wrap up the assembler side by moving the obvious unnecessary functionality to the Forth side and doing other tweaks that it needs.

And, because I can do studying, I proceed with that now..

And then I didn’t. I watched three movies with commercials from the TV that I didn’t even want to see. It was my worst enemy again - I call it the Media Psychosis, the inability to shut the damned thing off or close the browser tabs or let go of the printed newspaper before I’ve fixed its every typo. The battle continues, it seems. But I did like The Fugitive - exciting action, and I did learn for the first time that the popular children’s game Trouble is known as Kimble in Finland after the main character Kimble. It all makes sense now. Why else would they call a game Kimble.

I need to work on my problems, but it’s been like ths for so long that I’ve mostly learned to cope with it. Accept the wasted time, and move on, as soon as the curse gets lifted each time. Thinking positive: It must be at least 5 years since I last touched Nethack (and lost a few weeks by doing so).

Back to the project. After sleeping, I still think it is better to defer the larger coding work after the next Chapter is read.

Situation after the quick 3 hours of looking at code, but mostly procrastinating and rather unfocused:

3129 packtest.obfpoc.gz nFIXMEs: 48 #TODOs: 58

2020-12-12T12:52 (1 hour 51 total)

Chapter five: Implementation: Elements of Forth Style

Notes from the read

So, the first principle is to organize Forth code as a good book. From “the elementary” via “the intermediate” to “the advanced”, maintaining division to chapters, sections, subsections, and table of contents as the map for the reader. Decomposition vs. composition as food for thought. Brodie suggests use of a hierarchical structure, as in a book.

In 1983 code used to be organized in screens of 1K units of mass storage. Understandable. So the basic recipe for structure is screens, lexicons, chapters, load screens. Understandable, from a 1983 point of view. Hierarchic structure was made with a screen that LOADs screens that possibly LOAD more screens recursively. Nice… LOAD and THRU are still in the standard as of 2020, in “The optional Block word set”… Current wording is “block: 1024 characters of data on mass storage, designated by a block number.” (https://forth-standard.org/standard/block)

Yet - organization of source using 1K units is a little bit awkward these days. My system will not benefit from actually using blocks. But still, I’ll think about the timeless beauty of hierarchical structure as a possible universal virtue. Works for books - why not code. Decomposition is part of what we teach these days, too, of course. So… no news here, actually.

accordance to current standard, by semantics given by the optional block word set. That’s nice to know. I’ll be doing exactly that in my

seems… Also, the relative screen-reference word FH (“from-here”) is not in the current specification. Aha, but Brodie did already give a definition of it as “his own” in his Appendix C. Fair enough - it uses the standard BLK and I see no reason why it wouldn’t work today with standard block words, if need be.

The main point seems to be: arrange hierarchically. In the 1983 world, there is an unfortunate lack of semantics in the arbitrary screen/block numbers needed in the organization. We’ve seen a couple of technological advancements since then and the film Back to the Future already in 1985… I was six when the movie came out - I could read, but I had never heard about computers. I read only Donald Duck comics in Finnish and played with Legos. Oh, I must now get back from the future - focus on reading more about the past…

Ah, yes FH is definitely Brodie's own version as opposed to more common ways using +LOAD and +THRU. All of this is very historical. What is to be taken home, though, I guess, is the way that Forth enables one to define an "own way" of doing a thing.

An interesting note by Brodie: "Some programmers begin each chapter with a dummy word; e.g., : VIDEO-IO ;" - for FORGETting things, yes, but I think I might find some use for this in my current obfuscator logic, too. I think this is yet another piece of reassurance about the wonderful wild, free, and creative world-view that has separated Forth from contemporary alternatives back in the days… maybe today, as well?

Brodie continues with more and Moore [funny pun] about the hierarchical organization and documentation - or not so hierarchical organization. The take-home message is something like "it depends" once again. Quickly skim through things obviously relevant only prior to Spaceballs and Ghostbusters, and completely outdated long before Matrix came out.

Brodie's worry in 1983 was: ' "Infinite-length" files allow sloppy, disorganized thinking and bad factoring. Definitions become longer without the discipline imposed by the 1K block boundaries. The tendency becomes to write a 20K file, or worse: a 20K definition.' I think this has become canon. The main thing is to not be sloppy. And I think that sometimes breaking any rule is the proper thing to do.

Disk partitioning as a question related to source code structure - a very interesting historical note. A couple of years later I was 8 and I got an Atari ST with double-sided floppy disks, and never thought that disk partitioning with a few text files could have been an issue just a few years earler. The same goes with prints of source and slipping "triads" in binders. What can be taken home still today is the proposed way of handling the issue: "These standards must be set by each shop, or department, or individual programmer, depending on the nature of the work." And it all boils down to "It depends" as far as I can synthesize.

The story about "electives" makes me smile as I am creating a system that is all about having the bare minimum of anything. The philosophy of resource-restricted Forth is the same as the philosophy of making demoscene intros. Well, that is the reason why I got interested in Forth in the first place, thanks to my colleague Jonne having some experience with Forth and also the Thinking Forth book.

"Spacing and indentation are essential for readability." "Be consistent." "Phrasing is a subjective art; I've yet to see a useful set of formal rules. Simply strive for readability."

Spectacularly little news here, from the 2020 vantage point. Practice is practice, and we try to be good in our style, no matter the language or platform.

Comment conventions - again something that has become the norm in documentation comments of any platform… As for Forth, the stack comments have been pretty much codified in the standard text. And other comment types as well. So, as of 2020, use the currently codified style and basically skip this part of Thinking Forth…

Are they still using “data-structure comments” as described by Brodie here? If yes, I should follow suite.

In general, verbose commenting is encouraged. But with the twist that too much commenting can be counter-productive. Again, it depends.

The wording “narrative comment” is a descriptive one…

On page 161, there is a very nice quote-worthy phrasing: “Skillfully written Forth code is like poetry, containing precise meaning that both programmer and machine can easily read. Your goal should be to write code that does not need commenting, even if you choose to comment it. Design your application so that the code, not the comments, conveys the meaning.” In a way, this could be an ideal in all coding? And I think this is part of our canon still in 2020. But this is one of the difficult things to remember and do in everyday practice?

Tip 5.14 boils that down even more: “The most-accurate, least-expensive documentation is self-documenting code.”

And.. yes.. at the end.. it all depends.

Now, some sleep. I’m at page 163, subsection “Choosing names: The Art” next. Hmm.. only 7 pages to go in this Chapter. But the information density has been quite sparse from the 2020 perspective so far in this chapter..

2020-12-13T03:28 (3 hours 54 total)

Names are important. Brodie’s tips: “Choose names that work in phrases. Faced with a definition you don’t know what to call, think about how the word will be used in contex.” “Name the what instead of the how”.

I need to revise my code according to the nice tips about brevity and crispness.

And - I’m probably gonnna want VOCABULARY and friends in my system to get a sense of a namespace.

Hehe, I love the comment “Forth was created and refined over many years by people who used it as a means to an end. At that time, it was neither reasonable nor possible to impose naming standards on a tool that was still growing and evolving. Had Forth been designed by committee, we would not love it so.” This paragraph made me smile after implementing the couple of hundred standard words defined by a committee in 2020. But, I’m getting the underlying ideas about “it depends” and “just get the job done”. Strong ideas.

That said, keep Brodie’s Appendix E under the pillow when coding Forth. [No, as an afterthought: Forth, Inc.’s handbook contains more recent style guides; that one under the pillow, not Brodie anymore]

Reading pace: Pages 135-170 (36 pages; some blank; no exercises) 5 hours.

2020-12-13T23:28 (2 hours 56 total)

Take-home from Chapter 5

I think the main message of the chapter was: Make your code readable.

I couldn't help but read this in light of "Literate programming" on which Knuth published at 1984, one year after Thinking Forth. Of course, the lexicon idea of Forth enables a few extra levels on top of combining macros and natural language.

Not much news in this chapter. Of course I want to make my code readable and use all the tricks I can, developed before or after the 1980s. I'm gonna organize my system like a book, and comment carefully so that the story can be followed from pre-requisites named in the "Introduction"...

Again tired and incapable of programming. So I'm gonna take a sneak peek in the next Chapter and do the coding after some sleeps.

Reading pace: Pages 171-177 (6 pages; some blank; no exercises) 1 hour. Reading should continue from page 178 later. But first, I have a task to accomplish with code.

2020-12-14T00:42 (1 hour 57 total)

Code some more

Ok, coding spree starts. Goal:

- Finalize the assembler side. Or swallow the disappointment at 20 hours if that wasn't enough (at 77 hours into this episode of the travel).

There. To do or to fail - fast enough.

2020-12-14T00:42 (1 hour 57 total)

Well, well, the calendar jumped almost two weeks. Time really flies with the 12-hour working days and all this Christmas time. Finally, some hobby programming time again. Pick up the pieces from two weeks ago.

2020-12-26T23:12 (1 hour 58 total)

Some more hours - tedious re-cycling of IO from asm to Forth.. And not so focused. Intermixed with holiday activities. Once again, I'm counting all hours of trying to focus but not able to.

2020-12-27T17:00 (6 hours 64 total)

Getting the IO-stuff away from the asm side. Measures got a bit smaller again by just cleaning it and not creating more mess:

3042 packtest.obfpoc.gz nFIXMEs: 47 #TODOs: 52

Some of the remaining notes on the assmebler side seem to involve some experimental benchmarking of different implementation versions, so it will take even more time to deal with those than it took to remove the IO. And.. well, the old truth.. can't do it tired. Sleep, and wait for a brighter hour.

Commit some of these notes. [Note to self at 2021-01-24: semi-checked language up to this point. It could always be better, but shall suffice for now. There's no time for more language studies now. Continue from here onwards after some sleep..]

I gave myself 20 hours to finalize the assembler part, or fail. 10 hours out of those are left at the moment. We'll see how it goes. It might be useful to "fail" even earlier if the changes would actually be easier and faster to do in a later stage, with more of the obfuscator and maybe even the actual artistic code in place.

2020-12-28T02:22 (3 hours 67 total)

Revised the :NONAME stuff. It became a nice one-liner in Forth, and the assembly code just got much cleaner again.

2020-12-28T14:20 (2 hours 69 total)

Well, I've done some looking at the project today, on and off, and making small commits.. I have no idea how many hours I could or should mark for the time. Perhaps I need to err to the conservative side and mark all the hours, even though most of it was unfocused and inefficient.

2995 packtest.obfpoc.gz nFIXMEs: 44 #TODOs: 47

I make some deliberate "false starts" here to sample the way that the size might behave when different variations were used. It is just a sample of the present situation with the beep triangle and quite random words. But, it's going to be interesting, and it will give the best answer currently possible to one of the remaining wonderings in my original comments on the assembly side.

Size with copying code, manual NEXT, absolute addressing: 2995 Size with referencing code, manual NEXT, absolute addressing, abs calls: 2993 (unpacked 5828) Size with referencing code, manual NEXT, absolute addressing, rel calls: 2995 (unpacked 5777) Size with referencing code, manual NEXT, relative addressing, rel calls: 3057 (unpacked 5743) Size with referencing code, manual NEXT, mixed addressing, abs calls: 3050 (unpacked 5802)

Size with copying code, automatic NEXT, absolute addressing, abs calls: 2960 (unpacked 5568)

Even though this was just a sample, it reassured me that the earlier choices and ideas were good ones. There is this less obvious but hiddenly intuitive trend that repetitive data compresses better than data with more variance, even if the original file size is smaller. Entropy is the real enemy. Repetition is the friend. Also, it doesn't come as a surprise that deleting a lot of stuff reduces size if it doesn't come at a cost of adding a lot elsewhere - even if deleting repetitive stuff. Less is less, and let's believe it. The clear winner strategy, based on this single sample, is to:

- use absolute addresses for everything; well-packedness of any repeated uses outweighs the cost of each address being long by itself. [A later note: even better could be addresses relative to constant base register? A note from the following year: Yes. Done now.]
- automatically add the few bytes of code that always comes in the end of a word; necessitates re-writing of code to a new location, and thus supports compact representation that is programmatically unwrapped.

As of yet, I have found no counter-evidence. Yet, I postpone the final decision and code change to a later time. Why? I think it is a little bit brutal to write hard-coded bytes into an exe. I think I just want to keep my algorithm beautiful for just a few more hours.. This change can be done any time, and it will take 5 minutes. I'll do it - just not yet. Why do I hesitate? Isn't it commonplace and elegant to JIT things all the time - why do I hesitate on a mere 0x20ffad48 so much?

Interesting question.. not really related to programming but to the inner self and the human condition. So - think about that in some other diary… move on in this one. Decide: Leave this as a FIXME-note, so remember to come to it later.

Then the last bit of today's workings.. How much time did I spend? It is 12 hours since the last known stage. I've really been on holidays with my wife today: having fun, watching TV, cleaning the house, and being all social. But I've been looking at the laptop screen, too, multiple times every hour. Focus on nothing, but get some things obviously done.. record it how?

It's easier when there's the flow experience and a 110% focus on a task and no distractions. I'll just say this was 10 hours of project time - and make a note that I'm ashamed of how I had some 40% focus during all that time. I need to catch up, because I missed my own calendar deadline by 2 hours already!! I failed, and I didn't even fail fast.

So - I didn't finish with the assembly side yet. I got to something like 50%. Irritating. I'll come back to it later. Show must go on. I failed, slow. Now I steer back on the greater route. Read.

[A later note, 70 hours later when doing self-assessment: I disagree with all that momentary shame. Many things got done, and all these frustrations seem irrelevant and small when looking back afterwards. Something to remember, too. Going some hours beyond deadline is a normal thing, so the deadline should be set earlier than factually necessary.]

2020-12-28T02:33 (10 hours 79 total)

Chapter six: Factoring

Notes from the read

“an understanding of good factoring technique is perhaps the most important skill for a Forth programmer.”

You can factor out data, functions, code from control structures, control structures, etc.

Oh, dangerous-looking example: CONDITIONALLY with “R> DROP”. But this is Forth with a constant feeling of danger about?

How important an idea universally is Tip 6.5 “Keep definitions short”? It is part of the decompositional canon we teach. Is it always good? Worshippers of the short definition? Is it a cult?

Brodie quotes Moore: “A word should be a line long. That’s the target. [. . .] Short words give you a good feeling.”

[OCR error or original typo on p. 178 “out” should be “our”? on p. 179 “you timings” “your timings”?]

I think the actual lesson of Tip 6.6 comes in what is said after the tip itself: ‘Don’t let your ego take over with an “I can lick this!” attitude. Forth code should never feel uncomfortably complex. Factor!’ How can I remember this in my daily work? I mean.. let’s not think just Forth, let’s think just a thing X, not necessarily even code in any language. . . “X should never feel uncomfortably complex. Factor!” It is so easy to say. Is it easy to do when implementing X? For me, never.

The other facet in Tip 6.7 is worth remembering: “Factor at the point where a comment seems necessary”. Especially for Forth, the need of a stack comment would warrant a factor, says Brodie. Also for a newbie programmer? There would be quite a few small words then. But, well, this is what Moore would have suggested..

“reusable, testable subset” - good stuff, written when I was 4 years old . . . “Look for repetition of patterns.” Yes. “Be sure you can name what you factor.” Hard. Important. Something to think about and practice deliberately.

Oh. . . we’re getting back to “it depends” a lot.

Brodie says “Knowing when to hide information requires intuition and experience. Having made many design changes in your career, you’ll learn the hard way which things will be most likely to change in the future.”

Mostly quick readings. Not so much news, and so far doesn’t look like rocket science. 10 more pages in an hour, and I don’t even think I skipped too much.. But did I understand? That’s a question..

2020-12-29T03:40 (1 hour 80 total)

Aim to high mnemonic value. For example, “OVER + SWAP” can be better than “RANGE” because of its mnemonic value.

Very important tips towards the end of this chapter - canonical, once again, so no news in 2020, but timeless wisdom nevertheless. Quotes from Moore are insightful.

In the end, everything depends. Getting things done is the engineer’s goal. Quoting Brodie, “A good programmer continually tries to balance the expense of building-in changeability against the expense of changing things later if necessary. [. . .] These decisions take experience.”

It is OK to use clichés such as OVER + SWAP.

Take-home

The take-home message of Chapter 6 was summarized in the quotes. It depends. Get things done. Comes with experience. Go do and learn.

So. . . perhaps I should do as I’m told, and go try factoring in practice. . . But - looking at my project schedule - I’ll make a very fast look at the code, say, only as much as I can fit in today’s holiday schedule. Then I go back to reading - to get that thing done. . .

Reading pace: Pages 171-197 (26 pages; no exercises) 3 hours. This was very fast, perhaps too fast to really understand. Also, I did this part a little tired and with a two-week break in the middle of a chapter. Not very good for learning. Well, I’ll be perfect some other day.

2020-12-29T12:42 (1 hour 81 total)

Sneak preview of the next chapter, but not attempting to read with thought. Will do after today’s programmings. I’ll come back to page 198 then.

2020-12-29T13:55 (1 hour 82 total)

Code some more

2020-12-30T13:32 (1.5 hours 83.5 total)

2020-12-30T14:52 (0.5 hours 84 total)

Just some revisions, for a couple of hours. Metrics improved quickly with the momentum from the last coding session:

2977 packtest.obfpoc.gz nFIXMEs: 42 #TODOs: 52

Also, I think I cooked up my next exercise in the current themes: See if my beep triangle could work in Gforth. . . somehow. . . Would be a nice exercise in Forth and factoring.

But now. Stick to the grand plan, and read the next chapter.

Chapter seven: Handling data: Stacks and States

Notes from the read

The two sides of the implicit stack: potential for both elegance and unreadable nonsense. Wield it carefully. It's not a trampoline for acrobatics! Usually three items are manageable, as "this, that, and t'other".

I like the narrative of "We've already taken a wrong turn but we don't know it yet." It makes you ask how we could know about the wrong turn earlier, doesn't it… Unsurprisingly, the answer seems to be… with experience. Well, essentially. But it helps a beginner to have these rules of thumb, like "ROT ROT" or fourth stack item might be the sign of having taken a wrong turn some moments ago.

In this short project of mine, I've taken these wrong turns many times. I think it is essential to do the wrong turns first and feel the pain. Then, it will become possible to learn something about right turns.

So - look for the danger signs, and rethink/redesign early. I think this can be taken home to all engineering, not just Forth… As for the freshperson students, there is no substitute for just spending the hundreds and thousands of hours in doing, trying, and erring at first.

In the example of drawing a box, about using local variables: "If you're in a hurry, it would probably be best to take the easy way out." The easy way is local variables. After the book was written, I imagine in the 1990s, the optional word (LOCAL) has been introduced to enforce locality of local variable names and actual re-entrancy of definitions using locals. In the 1983 exposition, locals were just carefully used globals… Do I want to implement (LOCAL) in my system? Probably not - I've used too many hours on this project already… I suppose I'll just use the 1980s non-re-entrant fake-locals, if need be. Maybe…

The general advice is to not use PICK or ROLL, if possible. At least not to misuse them.

(Come back to page 204)

2020-12-31T10:45 (1.5 hours 85.5 total)

Look, it's a new year. Celebrations were comparatively mild, so the last man standing can read and understand some 1980s stuff. I just watched Jean-Michel Jarre's virtual new year concert live. Things and people from the 1970s are still very fine. I hope I can be as current in the 2050s in whatever I do as Jarre was in today's music performance. He even played the laser thing, and it didn't feel too last-millenium.

Global variables are bad, m'kay. Said Kogge in Brodie's interview long before we saw South Park.

Some interesting notes about novices in trust. Would require a longer thought, and that would be work, not hobby… steer away from it now.

Nevertheless, “I told you Ace, there were too many variables!” … fun in computing needs to be kept. I salute the 1980s Brodie.

Most things are reflections of “elegance” which was the first concept introduced in the book. Again, in Chapter 7, a quote from Moore begins with the thought “We should be writing for the reader.” Technical details may have become obsolete but these general truths not so much.

The BURY and EXHUME example is funny but not really too insightful. Or maybe I’m missing some point… And the HOLES business. It all seems to just remind us that it is worthwhile to think and design and not to be too stupid too often.

2021-01-01T03:20 (1.5 hours 87 total)

Application stacks seem to be encouraged. Why not.. Names with conceptual meaning should be preferred over “push” and “pop”.

State tables are recommended for interrelated state variables. Storage and change of state can be made in different ways. Aiming for readability is the main point again, I guess.

2021-01-01T18:55 (1 hour 88 total)

DOER/MAKE is Brodie’s own invention. Seems like a useful “pattern”. Here, I’ll make notes of also the Appendix B that contains the definition of DOER/MAKE, 5 pages in total.

The use of return stack in the DOER’s DOES> .. quite extreme.. can you really do that? Even in the 1980s? Wow. I’ll try to expand my head around all this dangerousness of Forth… The implementation of DOER/MAKE is given for four different Forth systems of the time. Heavily implementation-dependent. Some of the implementation differences of those four contemporary Forth systems can be seen in the given versions of DOER/MAKE.

Now… The basic version of this thing looks a lot like DEFER and friends of the current standard - so maybe some piece of Brodie’s visions here have become codified since?

Hmm… DEFER was not part of ANS Forth 1994, so it wasn’t in the first ten years that this happened… Now, after 26 years, we have its spec… at what stage did it get in the standard? Revisions of the proposal have been made as late as in 2004 (http://www.forth200x.org/deferred.html) Aha, and the table of contents on the main page (http://www.forth200x.org/) reveals that DEFER was "Accepted at the 2005 Forth200x meeting… I just wanted to know.. now I know, which is nice.

I think all this is good as examples and inspiration to the ways of the Forth. Looking at some newer and current Forth systems would be better today, though.

Take-home

This chapter was about managing data. General ideas might have lasting value. Key points seem to have been codified since the 1980s as (LOCAL) and DEFER and such. So… not really sure about what I can truly take home from this, through the time machine interface. Key ideas, I suppose, like "it depends - strive for elegance - you'll need practice".

What? Only one more chapter and a short epilogue remains to be read? It looks like I'll be able to complete the reading task well within the originally planned 5 ECTS episode? Yes. Cool. I may even need to extend the goal a bit. No worries… there are always more things to do and learn after an official course ends.

In this Chapter, there were many references to Chapter 9 of Starting Forth. I'm probably going to want to revisit that one more time. Also, for post-1980s viewpoint, maybe check http://www.figuk.plus.com/build/heart.htm at some point.

Reading pace: Pages 198-226 (28 pages + 5 in Appendix) done in 7 hours

2021-01-02T03:15 (3 hours 91 total)

Code some more

My next task:

- Learn how Gforth is supposed to be hooked to C libraries and implement the proof-of-concept beep triangle with Gforth.
- To keep this exercise manageable and useful, just re-create audio and visuals in Gforth - don't yet go to the "next level" exercise of creating an interchangeable compatibility layer / lexicon that can factually run the same application code on top of "any" Forth-system below. That is the goal, but it cannot happen immediately. Be patient. Small steps. Do as you preach, man…

After such an exercise, I'll have a proper perspective on making the interface levels of platform dependence and independence that I wish to achieve.

Started by copy-paste. Next up: learn and modify.

2021-01-02T14:57 (2 hours 93 total)

The SDL window is up in gforth.. so.. steady pace in getting this exercise done. Learning to use the C interface of gforth, yes.. I'll think about my own tiny interface later.

2021-01-02T19:45 (2.5 hours 95.5 total)

Window, update, and some of the other things seem to work.. some output looks almost correct… audio doesn't work, and the graphics is mostly very buggy.

But… there is something of a promise that it will start working after some debugging.

2021-01-02T20:35 (1 hour 96.5 total)

Diving into OpenGL header files and versions once again.. Legacy versions have some niceties for small code size, so I use those today, but I should be learning the newer things to keep current professionally, or Vulkan for that matter.. I'll put these learnings on the soon-after-this-project list of things to learn.

2021-01-03T01:40 (0.5 hours 97 total)

Graphics is fixed now. Debugging the audio (hangs if trying to use it..)

2021-01-03T02:44 (1 hour 98 total)

Audio now works, too. The exercise got done in less than 8 hours, within two calendar days. I learned how to link and call C library functions from Gforth, which, I hope, makes me better equipped to make a final decision about my own "graphics & audio lexicon" for tiny executables.

I did catch some bugs and omissions in earlier work. So, this time I got a bit of a setback in the measurables, but it's all working towards a better final outcome:

2986 packtest.obfpoc.gz nFIXMEs: 51 #TODOs: 58

And… qualitatively, I would call it a major leap forward that now the unobfuscated original source of the Proof-of-concept version works both on my own system and on Gforth.

When I come back to programming, the next exercise is of course the "next level" one:

- Factor out the differences between gforth and my own forth - make a tentative lexicon that allows the same application code to run on both. Try to make it so that (in theory) any other system could be running the stuff.
- After that, I want to keep gforth and my own to be interchangeable platforms that can both run the development / debug version of my audiovisuals at all times. To sort of prove the point that my demos are made in "standard Forth" so that I can go all Elvis about it :).
- After that, back to the obfuscator and minification tricks.
- And.. finally.. to have some bit of fun and creativity with the toy? The actual reason of all this.. finally, finally, get to the fun part!! That may have to be taken to the next episode because the magical limit of 5 ECTS credits is soon upon me once again.

But now.. The latest practical exercise is done, grade 5/5 for this one. Now back to reading the final part of Thinking Forth.

2021-01-03T03:45 (1 hour 99 total)

Chapter eight: Minimizing Control Structures

Notes from the read

Quoteworthy material, this time from Moore in an interview snippet: ‘Every conditional should cause you to ask, “What am I doing wrong?” ’. An ATM example of why conditionals are hard to understand.

Then we enter the Valley of It Depends. But with nice basic examples.

2021-01-03T04:20 (0.5 hours 99.5 total)

A case against state-smart words such as FIG definition of dot-quote vs. 1983 Standard version of separated dot-quote and dot-paren. Somehow I imagine that the discussion has been going on for 40 years because the state-smartness is already such a familiar concept from the Internet forums for a newbie like me. Well, the compiler/interpreter is a STATEful engine by definition, so some state-smartness will become familiar in implementing one… and it is a hard nut to crack at first. So, I see a lot of point in trying to avoid smart words and aim for dumb ones. And then… some words like the MAKE presented by Brodie work well state-smart, so once again it depends.

Use “else if” instead of “then if” for mutually exclusive decisions. Combine logicals to one boolean instead of nested ifs, unless there are different costs from their computation time. Forth’s boolean operators don’t short-circuit. Thus, sometimes ugly and repetitive code is necessary.

Tip 8.7 contains a quote-worthy opening, in my opinion: “The most elegant code is that which most closely matches the problem.”

2021-01-03T14:00 (1.5 hours 101 total)

Oh, there has been a Charles E. Eaker who proposed the CASE OF ENDOF ENDCASE words in an article called “Just in Case”. Wonderful history, yes. And quite an interesting opinion from Brodie that the case statement is not sufficiently univeral to warrant inclusion in all systems (“an elegant solution to a misguided problem”). Well, even today we find it in the core extensions instead of core.

Choose looping constructs, and create your own ones. This will be one thing that I’ve been really looking forward to. I hope there will be good examples of how and when to do such artistry. (An afterthought after reading the whole chapter: was this thing in fact the part about EXIT and R> DROP and LEAP … it wasn’t super clear to me on the first reading…)

Approaching the tricks section. Indeed, in a two’s complement system there can be “: S>D DUP 0< ;” how anarchistically cool.. And then there is the examples from Moore about “t 0< 1 AND” … I think these little things will be fun and possibly useful for size minimization. “Hybrid values” Brodie calls these values-slash-booleans. At least they should be fun to use and dazing enough to the unknowledgeable bystander. Reminds me of “a|0” flooring and similar minification tricks in Javascript .. or “push rax; pop rbx” instead of “mov rbx, rax” to save a byte of size. Brodie does discuss the nature of such “tricks” thoroughly. Again, it’s like “know thy audience”, and document assumptions clearly.

2021-01-03T17:15 (1 hour 102 total)

Decision tables and all that. Rather canonical these days… Math look-up tables not so relevant today, perhaps. Philosophy of engineering and tricks timeless.

Add complexity at the bottom! Smarter apples, not pickers or sellers…

The case against checking arguments within-system… I disagree. Assert is friend, and tests with some coverage good to have. Trade-off with time-to-market is of course a reality. But to train for a paranoid mindset is not necessarily unhealthy.

Moore: “Instead of trying to get rid of conditionals, you’re best to question the underlying theory that led to the conditionals.”

2021-01-04T00:25 (1.5 hours 103.5 total)

Side-tracked to the Gforth manual for 2 hours. Not a bad round-trip at all. Very instructional, but not related to current goal. Focus, dude.

2021-01-04T09:55 (2 hours 105.5 total)

Brodie on using R> DROP: “Fooling with the return stack is like playing with fire. You can get burned. But how convenient it is to have fire.” Funny and insightful analogue. I try to use this kind of wordings in my teaching, too… Go on and play with fire - but don’t get burned… Personally, I would hesitate very much to play with things like R> DROP and LEAP which might be either R> DROP or R> R> 2DROP or something different, depending on the system..

But then again.. I’m in the hobby of building executables that break every rule in the house to be tiny: Javascript packed in PNG, Shell scripts dropping binaries to /tmp/ for execution, ELF binaries reporting fake file sizes, calling libraries with hashed function names, … perhaps I should view the sinfulness of R> DROP or other dependence on some details of my own Forth implementation in that context. And just let things burn when it’s fun that way!

Regarding simplicity, Brodie quotes Moore on language: “It depends on your intention: simplicity, or emulation of English. English is not such a superb language that we should follow it slavishly.” Now, this is interesting philosophy in programming and creating systems (for multicultural audiences). The thought needs to be abstracted by substituting any natural language or the phrase “a natural language”:

(Moore re-phrased) Mind your intention. Is it simplicity, or the emulation of a natural language. The natural language is not so superb that we should follow it slavishly.

Then, the plot thickens and gets fuzzier when the intention *is* to emulate a natural language of a user from this or that culture... But to think of just programming - it is easiest to do in the syntactic rules of the programming language, not necessarily the native language of the programmer. And the plot thickens in other directions - which programming language best matches the programming task at hand, and should we use that one. This won't be a cliffhanger though, I think.. I guess the right answer is the cheapest one, i.e., to use whatever language the available staff currently knows best. But, I still think that as teachers of university level programming, we need to train our students in diverse languages, so they at least know cultures other than those that have Java or C in their names.

That's it then - I've read the Chapters of Thinking Forth. There is still the Epilogue, but I'll have a break first.

2021-01-04T11:36 (1.5 hours 107 total)

Take-home

The final chapter was about ways to not do an IF when possible. Harness the power of the language being used, Forth in this case. I think this is once again mostly what we already take for granted in 2020. Most of this stuff might have been new and revolutionary in the 1980s, but then it was a time when the whole software engineering field was half the age it is now. If not infancy anymore, it could have been its wild young adulthood?

I'm starting to think about the overall take-home message of the whole book. Mostly it was about "do what needs to be done", "it depends", "tricks are OK". There might be more, but I can't think of it right now. This is one interesting thing by the way... it is very superficial to try and "take something home" immediately after an experience. I think it needs time and further experiences (like doing things in practice) and even just sleeping and doing other things in life... to reveal what should be taken home from the earlier experience. I refrain from trying to do the impossible right now, when I've just read a whole lot of new stuff. Initial thoughts would be superficial and forced - thereby futile.

Reading pace: Pages 227-260 (33 pages, one small exercise, some auxiliary readings) done in 9 hours.

2021-01-04T12:25 (1 hour 108 total)

Further readings

Now that I'm done with Thinking Forth, what would be the next readings? I'm thinking about the following ones:

- Forth Programmer's Handbook by Conklin and Rather and the technical staff of FORTH, Inc., 2007 (3rd edition)

I skimmed through this one already, and it looks like good readings for the modern day. Refer to this one for modern conventions.

Leo Wong's "Do not shun Scylla by falling into Charybdis" and other readability rules are a good continuation and dialogue after reading Brodie's 1980s expositions..

From the bibliography of the last mentioned (I haven't taken a look inside yet):

- Pelc, Stephen. Programming Forth. Southampton, England: Microprocessor Engineering Ltd., 2005.

  "A good modern text by the Managing Director of one of the leading Forth vendors. Downloadable from www.mpeforth.com"

- Rather, E.D. Forth Application Techniques. Hawthorne, CA: FORTH, Inc., 2003.

  "An introductory text on Forth used for courses at FORTH, Inc. Includes many examples and problem sets. An ideal workbook for the beginning Forth programmer."

I suppose that the Gforth documentation might be quite up-to-date, maybe:

- The Gforth tutorials: https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Tutorial.html
- The Gforth Introduction: https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Introduction.html

I browsed through many of these pages, and they do have a lot of useful insight and ideas concisely written.

The SwiftForth manual is likely to have another up-to-date view on how to work with a 2020s Forth system.

The most up-to-date and into-the-future things seem to be happening all the time at the standard website. I haven't dared to look much into the mailing list / news / whatever channel the community might be using for discussions.. At some point, perhaps. But I must remember how limited a resource time is.

I did read the epilogue already, and made sure no page remains unturned in Thinking Forth. I still have 24 hours left of this episode of my project - that is almost a whole credit point worth. I'll save the last 5 hours to documentation and transition to the next episode. Now, I'll use the remaining 19 hours as I laid out yesterday:

- Factor out the differences between gforth and my own forth - make a tentative lexicon that allows the same application code to run on both. Try to make it so that (in theory) any other system could be running the stuff. (Got done in just a couple of hours, thanks to holidays and being able to focus on the task)

- After that, I want to keep gforth and my own to be interchangeable platforms that can both run the development / debug version of my audiovisuals at all times. To sort of prove the point that my demos are made in “standard Forth” so that I can go all Elvis about it :). (Corollary of the first point).
- After that, back to the obfuscator and minification tricks. (eventually I spent all the remaining hours and a few extra on this matter, with results that make me very happy.)
- And.. finally.. to have some bit of fun and creativity with the toy? The actual reason of all this.. finally, finally, get to the fun part!! That may have to be taken to the next episode because the magical limit of 5 ECTS credits is soon upon me once again. (Deferred to the next episode, with no burden to the heart)

2021-01-04T15:15 (3 hours 111 total)

Code some more

Hmm… suddenly it is hard to turn the head from reading text to modifying code.

2021-01-04T16:24 (1 hour 112 total)

Some distracted and unfocused hours, but it’s been a good day.. Float and float-ext words updated. Made my system similar to Gforth.

2021-01-04T23:41 (1 hour 113 total)

Better. Very little sleep, but no distractions. Focused time, and got the unification done. Same development version of the “artwork application” can now be run either on my system or on Gforth with no quirky ifdefs. So.. I’m as close to standard as I ever wanted to be with the unobfuscated code…

Next up: Just a little bit of looking at the obfuscation thingy. Then to alter the beep triangle and try to be creative within standard-ish application code.

Starting situation: 3024 packtest.obfpoc.gz nFIXMEs: 44 nTODOs: 52

2021-01-05T12:28 (5.5 hours 118.5 total)

Then I did open the email inbox of the work, after two weeks of not opening any boxes. Short are the free periods of a working person.

Yet, the special year of 2020 has strangely proved it possible for me to be able to do some of these nice hobbies every now and then. Things are good. I’ll be able to work with this again, at some point soon. But it will be after we start the year at the uni.

2021-01-05T23:20 (1.5 hours 120 total)

Two hours of nice evening coding in-between one hell of a week at work. Now, off to sleep… with a feeling of achievement… I have an actual self-executing binary in well under 3k. Of course it is still just the beep triangle, but now it comes with more than 1k of pure artistic space. All the boilerplate is there. And I have a feeling that there are some quite easy size optimizations available to get more margins.

2833 packtest.runner nFIXMEs: 42 nTODOs: 51

2021-01-12T23:55 (2 hours 122 total)

Ok… so now to sleep. (Ended up just examining the current gzip with gzthermal; surely, there are "easy" optimizations available, evident from the listings, but the easy solutions will be dependent on some hardcoded style/idioms used in source.. I'll find out soon enough. Now… to sleep.)

2021-01-13T00:50 (1 hour 123 total)

Wow - very tired and unfocused hours after a week at work. Yet, some progress with the measurables:

2775 packtest.runner nFIXMEs: 44 nTODOs: 51

Also, I think I'm beginning to understand more of the idea of the "lexicon" and the one-line word ideology spelled out in Brodie's classic. I think. Not sure. Just a thought that I might…

Only 4 hours remaining before wrap up and self-assessment of this episode. Let's see what can be done in such little time…

2021-01-16T02:19 (3 hours 126 total)

A 15-minute hack to remove unused array creation, done during a lunch break…

2735 packtest.runner nFIXMEs: 43 nTODOs: 51

It hurts to register a whole hour for this 15-minute hack, but I suppose I've been thinking about this subconsciously in recent days, so it sort of has been time spent in this project..

2021-01-20T12:58 (1 hour 127 total)

Obfuscator truncates most common words to one-char-codes with mnemonic value.

2711 packtest.runner nFIXMEs: 43 nTODOs: 51

We go without a window name - as we will in a demoscene competition.

2675 packtest.runner

Only two more hours before the end of this episode. I'm happy to be under 2.7k with a beep triangle. That leaves at least more than 1k worth of creative space. Nice things can fit in 1.3k. Maybe a little bit disappointing.. but of course there are still some known tricks not used, so it is not the final restricting limit.

2021-01-23T01:10 (1 hour 128 total)

Used some of Brodie's tips. 25 bytes came off in 30 minutes of programming. By extrapolation, I would be at 0 bytes in only 40 hours, but reality doesn't work that way… Every 25 bytes saving in 30 minutes is a win and feels like an achievement. Gotta love this hobby.

2650 packtest.runner nFIXMEs: 43 nTODOs: 50

Then I spent too much time admiring the results. Should've just gone forward. Didn't. Live and learn. Only one hour left in this episode before wrapping it up.

2021-01-23T02:05 (1 hour 129 total)

And that's it then. Pretty tired after the week at work, so not many improvements. I did my best, so I'm happy. Time is up, so let's go to the wrap-up and the change of reel.

2021-01-23T03:07 (1 hour 130 total)

(—> Go to epilogue no later than at 130 project hours! … was my note to self here …)

I missed it by two hours. I just couldn't stop the movement begun last night. Yet, 80 bytes in 2 hours while watching TV and having some beer was good and relaxing activity for a Saturday. But now - focus, do the wrap up. Final status:

2570 packtest.runner nFIXMEs: 43 nTODOs: 51

2021-01-23T23:08 (2 hours 132 total)

Epilogue on epilogue: Effect of Forth on my thinking?

The epilogue of Thinking Forth gives three small testimonials of how exposure to the Forth language has affected other aspects of the work of professionals. The reported effects revolve around factoring, modularization, and simplification. Now, after reading the whole story, I maintain that these are universal virtues that we wish our CS students to start learning from day one, no matter the language. Yet, it is possible that these virtues surface in Forth more than in other languages, so Forth could be a nice playground for learning and exercising "lean thinking".

But now, the key question is: What kind of effect has Forth had on my own thinking, now that I have spent 20 ECTS credit points worth of time tinkering with it, building my own Forth system, and reading two classic textbooks, a standard, and half of the Internet?

I think I may have developed a little bit sharper an eye for spotting factors and become a tiny bit less lazy to take the factor out as its own definition. At the moment, I don't have time for other projects, so it is hard to say if the thinking carries over to things other than Forth. I think that all this 15 years of academic experience and a defendable doctoral thesis has already built a sense of "do what needs to be done", "just make it work", and "it depends". So I think the ultimate effect of all these Forth things might be marginal in the bigger picture. But it remains to be seen - I think the practical application programming will provide some more insight, and that lies ahead of me still.

At the end, I still think that most of Brodie's deeper insights and "Tips" are part of the present-day canon if we look past the language of Forth. We try to teach much of the same virtues in our classrooms in order to coach good programmers for the industry in any programming language.

Interesting and inspiring as it is, I don't think that Thinking Forth is something that a present-day student should spend much time on - unless they want to understand Forth. So, it didn't become a top reading recommendation for classes. I was sort of expecting even more of systems design philosophy that would exceed the confines of just Forth.

In Forth itself, though, there is a certain philosophy involved, and it was discussed thoroughly. I guess this facet might be one of the reasons why my colleague once recommended Thinking Forth as a book to read, regardless of specific interest in Forth or not. I have to admit that I haven't read the newer programming textbooks, like anything after Stroustrup's C++98 treatise. So I don't know if people are still writing about design and problem solving philosophy in a similar way. I hope they do. I hope they have funny concept illustrations like those drawn by Brodie. And yes, I should read some new books, too. These old ones were very nice, but, well, it is not Atari ST time anymore…

2021-01-24T00:16 (1 hour 133 total)

Wrapping up and moving on

Recap of the whole trip through Thinking Forth

At this point, I find it natural to go through my notes about this episode, and recapitulate the learnings. I'll do some language checking to the best of my capacity, but I'll also try to maintain the schedule of 2 hours remaining. It will not be enough to do a careful read of this much material. We'll see how it goes..

It goes like this: It took 3 hours to read the first half. I'm over the allocated hours. But I think it is worthwhile to do this recap, so I'll press on for a bit more.

2021-01-24T02:00 (2 hours 135 total)

Read about half of the notes. Quite a lot of work.. which I knew, and should have started a couple of hours earlier.

Read up to new year's events.

Read all the notes, and caught up with this line here. Took 2 more hours. I transferred the remaining time stamp and hour counts to the next episode in order to be consistent with the strict 135 hour slots. It took a total of 5 hours extra to revise notes. I consider it redundant to this episode, because this was already at a very acceptable state before the extra work. It becomes the first 5 hours of the next part.

Self-assessment

Quick self-assessment of this episode, by goal:

- I set out to read Thinking Forth, and I did, 5/5
- Actual application programming, having fun finally. I didn't, but instead I think I made the best decision possible by deferring this to the final "Practical" episode. 5/5 on being able to prioritize dynamically.
- Dependency-graph solution for identifying unused words. Done, but a hack. Works. 3/5
- Reduction of the number of small todo-notes goes without saying. Went down from 59 FIXME notes to 43. Success. 3/5

Overall, this episode met all its goals, which probably means that even the goal setting was successful. I ran over the allotted time budget by 5 hours (4%) which is the only thing that brings my numerical self-assessment down to grade 4/5 from the maximum.

The notes on reading pace was an important benchmark for any further work. I haven't done such timings since my freshman times 20 years ago, so that knowledge was a good byproduct of this thing.

Make a conclusion about how much to expect from a student in a number of hours? A lot! 135 hours is sooo much time!! We can really expect them to acquire both knowledge and skills in 135 hours. As long as their motivation and study techniques are up to speed, which they sometimes fail to be. But we need to attack the root causes of problems in motivation or study techniques.. our expectations (talking about my own house and colleagues here) per study hour are Very Fine presently!

The morning hours of hobby programming! A precious finding and a definite keeper for the future!

Note on my own note to myself, "more doing, less writing": I'm not entirely sure if that was a good note in the first place. I didn't follow the advice, as can be seen. But now, it felt very useful to read the notes afterwards, and I think it was so much easier to recapitulate the learnings. And I think that writing the notes made me think more carefully back at the time of studying. So... I think I shouldn't require less writing after all. "More doing" is a valid part, though. Doing is of essence. Focused doing.

Beep triangle came from 4032 bytes down to 2570. Massive reduction!

I'll tag the current version "POC-540h" in the git repository, to be consistent with the earlier tags.

Where to go from here

The creative part, of course!

Notes-to-self: Remember notes of 2020-12-07. Remember notes of 2020-12-10.

This episode closes here, and the next one begins.

Part 5 / Episode V: The Practical

(Learning diary of a hobby project.)

Transition between the previous episode and this one

Once again, I record hours spent in the "aftermath" of the previous part here. More or less accurately - the 6 hour total matches reality as a conservative estimate (less was certainly spent in reality).

2021-01-24T03:00 (1 hour 1 total)

2021-01-24T03:29 (0.5 hours 1.5 total)

2021-01-25T01:20 (1.5 hours 3 total)

2021-01-25T01:15 (2 hours 5 total)

2021-01-25T15:40 (1 hour 6 total)

I'll try to be finished more strictly within 135 hours in this episode. I intend it to be the last one. Maybe. Maybe not. We'll see.

Theoretically, this could the final episode that concludes a total of 25 ECTS credit points worth of my simulated self-study. The special significance of the particular number 25 is that in Finnish universites 25 ECTS is a very common size of what is called the Basic Studies of a subject. So, after this episode, I could imagine that I am finishing "basic studies in Forth" that had 3 introductory courses, 1 advanced theory-oriented course, and finally 1 capstone practical application course, each worth the usual 5 ECTS credits allotted to one course. Let us disregard the minor detail that university basic studies would expect no previous knowledge of anything necessary to understand the new subject. I may have possessed a lot of background knowledge on computer architecture, graphics, and math upon starting. Nevertheless, I think we always build upon *some* earlier knowledge, and the rest is pretty much connected to the time spent on the matter.

There may be compelling reasons for me to continue all the way up to 30 ECTS, though. 30 ECTS would be the size of the thesis project in my faculty, and also

I would end up with a total of 6 episodes, which sounds more complete than 5 - there couldn't be Star Wars episodes I to V without episode VI. The story would be totally incomplete. It may well be the situation after this fifth Forth episode of mine that a sixth one will be necessary.

Nevertheless, I want to achieve the original main goal of this whole project already within the 25 credits, 5 of which remain today.

Goals

The single goal in this episode is:

- Make a demoscene 4k intro that could go into a competition. Ready to be submitted in a compo system as an entry.

In this episode, I shall have no other goals. Specifically, I firmly forbid myself from getting side-tracked by the obvious:

- I should not go micro-optimizing anything. I have 1500 bytes here at the beginning, and I go with that.
- I should not try any of the structural change ideas for size optimization either. 1500 bytes. Go with that.
- I should not redesign the obfuscator. It is a terrible hack. Go with that.
- I should not think about any of the other nice future ideas I have made notes about. In fact, it would be best not to revisit any old notes until this episode is wrapped up.

The allowed exception is when an obviously more elegant solution to something would suddenly pop into my mind. But I should steer away from deliberately trying. This episode should be *all* about creativity. I have waited long enough to get here.

I hope I remember to look at this page every time I feel the urge to optimize. The urge will come many times, I'm sure.

Artistic plan

I have been thinking about this for some time. Now I write it down. I want:

- Aesthetics of the 1970s and 1980s computer graphics, as a homage to Forth origins. I'm thinking original Star Wars, Atari arcades, original Elite game. Discrete pixels, wireframes, black & white/green and fixed color palettes, dot matrices. Spinning cubes, starfields, plasma effects, sprite animations?.. Musicwise: chiptunes and arpeggios.. Kraftwerk, Jarre, Tangerine Dream, Vangelis, 3-channel sound chips…
- Allusions to lofi.. VHS, analog TV, home videos.. 8/16 bit single cores.. Out of focus, leaking color, blurry, interlace error, tearing, static/glitching

at times.. instahalf FPS when rendering one pixel too many… that sort.. which don't really happen these days anymore unless simulated.. See also things people do with PICO-8 nowadays..

- Modern take on everything, of course. I'm thinking "80s retro-futuristic" as in the Total Nuclear Annihilation pinball, Matrix movies, Stranger Things, Cobra Kai… Think LEDs in place of light bulbs, drone footage, and all that… Top notch today everything but portraying the years we yearn. Lofi meets hifi and so on.

Yep, writing this during lunchbreak, I'll get back to research now, but already listening to Scott Danesi's pinball soundtrack on repeat to remain inspired…

2021-01-26T11:55 (0.5 hours 6.5 total)

The wisdom is that constraints should not be thought of in early design, but here I play the "It depends" card. The packed exe size constraint is so rigid that I have to think about it from day one. So, the one rule to govern all my art:

- Keep it simple. Hint, suggest and allude.. fake if need be..

Music:

- FM synth.. darn easy to implement, and suits the retro theme
- Start from the simplistic Bomm-duh-duh-duh Pff-duh-duh-duh … suggest the "coolness" of TNA pinball (honest homage, it's a Very Cool Game all in all).

Graphics:

- I'm thinking a simulated 640x480 pixel framebuffer that is rendered in cool ways as texture in a raytrace/raymarch scene. Why not even many of those framebuffers, overlaid and blended in there. Wireframes, triangles, video walls, disco ball? I don't want to restrict my imagination before I get to try some alternatives and go where the flows takes it.

Drama:

- I really would like to have the string "' INTRO EXECUTE" in pixel font to show up in the beginning, with a cursor.. and "ok" appear right at the end, when music has gone to silence. To allude to the forthiness. Will be fake of course, and storage of font and text data may be a no-go restriction. We'll see about this nuance..
- Anyway.. start with a bit dull-looking framebuffer and lofi-only sound or noise. Go to the song with a flashy bang..

  Then dwell in the scene as long as it can be interesting, and go to outro. Aim to get a good 1.5 minutes of something interesting..

  I won't be dreaming of a second scene during this small project. If.. if.. but no, it is better to not dream too much at once. It is best to just start coding and creating. Audio up and running first.

2021-01-26T19:10 (0.5 hours 7 total)

Warming up

First steps with music

Yet again some nice coding time before going to work in the morning. It's not super optimal, but so far the best I can do to keep things rolling on the hobby side of life.

2021-01-28T06:55 (0.5 hours 7.5 total)

From a simple beep to an FM boing.. thinking about the synth / song lexicon. The wisdom about how much to design before just coding? I go with the other part of the same wisdom: Just make it work. All the code will be re-written later. Just play around with it, and learn about the domain.

2021-01-31T01:45 (1.5 hours 9 total)

Implemented F** to get frequency from MIDI note number… Obvious internal building blocks of a sequencer lexicon. So.. "it depends" and in this case I happen to know exactly what is needed, so I can just do the programming. Design will reveal itself, no need to make it excplicit yet.

2021-01-31T15:30 (1 hour 10 total)

Thinking about the song lexicon.

2021-01-31T22:30 (0.5 hours 10.5 total)

Some beats can now be generated by just commanding words for bass drum, snare drum, FM bass, and a step sequencer pause. Repeating a beat pattern is exactly a Forth DO LOOP structure. So, the main ideas are building up. It's nothing aesthetic yet, but it is not a mere beep anymore. So, the proof-of-concept is starting to shed its skin..

Now, go to work for this week.. If there is time in the evenings or mornings, start thinking about the graphics lexicon.

2021-02-01T10:39 (1.5 hours 12 total)

First steps with graphics

Ok. Now that beep-triangle became FMbass-triangle in only 5 hours, it is time to make it something other than a triangle.

Immediate observation: I haven't really done modern OpenGL enough to be able to just apply it in Forth. So.. first things first. Where are the tutorials? "Go to the source".. opengl.org and look at the links. Just some picks:

- http://www.opengl-tutorial.org/

  (OpenGL 3.3, Website CC-BY-NC-ND, codes WTF Public License)

  Goes through basic to intermediate things. Definitely usable as external course material in university because it would be NC use. Note-to-self: look deeper into this tutorial when lecturing the intermediate course next time..

- https://nicolbolas.github.io/oldtut/index.html

  Looks like a graphics textbook using OpenGL 3.3; I won't need the theory, I just need very simple usage examples.

  Note-to-self: look deeper into this, too, as possible external course material for my own students..

- https://learnopengl.com/ (by Joey de Vries, code examples CC BY-NC 4.0)

  Plenty of material, looks promising. In fact, this looks like a very good source from introductory to even advanced rendering methods. Definitely goes to the list of things to read and recommend to students!

- https://open.gl/introduction (codes CC0, content CC BY-SA 4.0)

  This was one of the first links I ended up surfing from opengl.org, and I got a couple of hours head start with this, so I think I'll just go along with this one for starters.

So… I'm up for the same old ride again, now with Modern "Core" OpenGL. Which ride? The one that starts with a triangle, then color, then texture, then… I'm gonna need stuff up to rendering in framebuffers and using those as textures in further steps..

2021-02-03T18:00 (3 hours 15 total)

Detour: A small self-study module on modern OpengGL API

No, I'll go where the river flows. Maybe I'll pick up opengl-tutorial.org because I haven't started yet. It seems robust, and carries further if time should allow… But… There's no indication of the people behind it. That is almost a reason for me to go with Joey de Vries…

Nah, I'll mix and match.. It is supposed to be the same story in different words anyway. Go with opengl-tutorial.org until a reason arises not to.

Learn to walk again - now in a modern style. Forget what I know about OpenGL (old versions), like the tutorial instructs me to.

Completed "Tutorial 1 (of any tutorial set): make a window". Of course I had a window from very early on, but now I tried some of the options to force a forward-compatible OpenGL context. Looks promising in that it shows blank screen for the current legacy triangle.. And I refactored, and cleaned up, and extended, and all the usual sort. Can't wait to get my hands on “Tutorial 2: actual triangle”, i.e., replace my glBegin(GL_TRIANGLES) with a core triangle.. But now, must eat, sleep, and go to work..

2021-02-03T21:20 (2 hours 17 total)

Went to the Registry to pick up the latest and greatest Specification. Always nice to check the truth when a tutorial says “don't worry about it yet, but this or that must be there because it is required. . . ” Oh, and there seems to be great wallpapers for decorating a nerd cave: https://www.khronos.org/files/opengl46-quick-reference-card.pdf

Yep.. and this opengl-tutorial.org says it right in the beginning of Tutorial 2: “you need this . . . but this is not very important”. So. . . Read OpenGL specification about these mystic VAOs.

From the spec: “The buffer objects that are to be used by the vertex stage of the GL are collected together to form a vertex array object. All state related to the definition of data used by the vertex processor is encapsulated in a vertex array object.” How hard was it to cite this. Ok, ok, . . . this is on page 349 of the current version, so I may have skipped a few details about what “all state” means. Maybe it's just me, but I really like to know where the specification is, and how it defines a thing - even if I don't fully understand the meaning. At least I know the terminology and number of terms that I need to catch up with.

Note to a fellow instructor: The shader creation in opengl-tutorial.org is a bit wordy compared to some other tutorial I peeked at earlier. Relies much on copy-pasting the example code at first..

2021-02-04T08:08 (3 hours 20 total)

Ok, it had to be finalized before going to work. Now I'm in rush to get there. . . But hey, 4 hours of morning coding, and I'm done with “Tutorial 2: Triangle”. Then, to “Tutorial 3 (of opengl-tutorial.org): Matrices”. But, I get to this one after at least a day at work.

2021-02-04T09:55 (1 hour 21 total)

I read too much on my lunch breaks anyway.. usually news from yle.fi (the Finnish Broadcasting Company) but this time I read the all-important matrix tutorial. opengl-tutorial.org is brilliant in selecting the points when to say “that's how it magically works, don't worry about it”. It is not to be read as a real introduction to computer graphics, but after doing enough homework with math with a more thorough source, it gives a narrow, straight, and short enough path through getting visuals on screen with C++ and OpenGL. Hmm.. But as for me. . . I'll need to design some Forth words for matrices in order to do matrices with it in the first place..

2021-02-04T13:21 (1 hour 22 total)

Obsessive coding and reading from the minute I got out of work today. Done with "Tutorial 3, matrices". Obviously I need to implement some further matrix stuff later on; now I just checked how to send uniforms to shaders in core OpenGL. No different from WebGL, which I've been using earlier, so no news in that tutorial. Also, I got rid of the old glRecti and glBegin stuff.. so I'm approaching 3.3 with my proof-of-concept. And the triangle rotates using a matrix built on the Forth side.

2021-02-04T19:30 (2 hours 24 total)

Quickly towards "Tutorial 4: A colored cube".. The spinning cube would be the A of the ABC of demoscene productions, right ;).

2021-02-04T22:20 (1 hour 25 total)

Note to self: This seems to be the first point in which the different tutorials start taking different routes around the matter. I have chosen my path for today's journey but I think I'll want to visit the other tutorials at some point to make an educated recommendation for students..

The colored box example is a bit artificial. I get the pedagogical point of showing some things that need an example around it. This must be how my students feel when they have to do all the dumber exercises with no real use. Nasty when it happens to yourself.. Should situate the learning, always.

Well, I made some random scales and rotates and Forth words for dot product and matrix multiplication.. probably all buggy as of now. But I feel like I'm learning. And progressing. Next, the coloring and the Z-buffer things. Probably going to make a matrix stack quite soon. It's all about leraning to walk in a modern way.

2021-02-05T01:15 (2 hours 27 total)

Done with Tutorial 4. But in trouble with perspective projection now.

[I'm doing these time logs manually, and here I found a two hour miscalculation. Corrected later, time added to the 2021-03-03 note]

2021-02-05T03:00 (2 hours 27 total)

Ooph, what a messy show.. keeping track of the float stack. I had almost forgotten the pain since tinkering with my 1k x87 synths. Pain remembered, problem solved, and a beautiful spinning cube!!

Now I'll still need to do the couple of exercises in the tutorial in order to learn about resetting the attributes on the fly.. So, change colors in runtime..

2021-02-05T20:35 (1 hour 28 total)

Oops.. Just realized that I haven't actually tested the obfuscated version for some time now… and it has stopped working. Nasty thing.. there has been so many changes.

2021-02-06T01:48 (2 hour 30 total)

Yep, the problem was with string contents being obfuscated as if they were normal Forth. It's messy, and took time. Now it rocks again after obfuscation. As a side effect, I got some more OpenGL error checking routines integrated. But spent 3 hours pretty much off-topic.

2021-02-06T13:30 (1 hour 31 total)

Time to stop playing with the unfinished toy, and come back to building it.. The old procrastination issue again. Focus! Tutorial 4 is finished with exercises, so start "Tutorial 5: A textured cube"!

2021-02-06T14:15 (1 hour 32 total)

Hmm… texture image needs 9 parameters, so I'll have to rethink my C interface a bit…

2021-02-06T17:05 (0.5 hours 32.5 total)

Meanwhile, very good restaurant food and catching up with friends. Later, back to this business.

2021-02-06T23:00 (0.5 hours 33 total)

Looking at the SysV ABI AMD64 supplement once again, and reverse-engineering gcc compiler output as example.. rethinking my own C interface.

2021-02-07T00:01 (1 hours 34 total)

I think I might have the possibility to make a 9 param call.. I'll try my luck with teximage.

2021-02-07T23:20 (1 hour 35 total)

Ok. The call code seems to br fine. Now I can call glTexImage2D and get a texture mapped spinning cube!!

2021-02-08T00:27 (1 hour 36 total)

Played a bit with the options, but moving on. I'm now through with Tutorial 5, texture mapping.

I went through "Tutorial 6, keyboard and mouse" very quickly. I won't be needing interactive movement in this project. The opengl-tutorial.org continues in it's nice "do this and it works" without much explanations. We'll have to remember that it's not a graphics course but a hands-on tutorial. And hands get on very fast with it. Also when everything is built from scratch with a new programming language.

Also a fast passage through "Tutorial 7, model loading". OBJ file loading pretty much like on the MIT OWC course I've been using.

Backface culling turned on.. it was mentioned in one of these tutorials..

2021-02-08T01:20 (1 hour 37 total)

Updated the Gforth version of the unobfuscated poc - seems to still work, and I intend to keep it that way until the end. Also checked that the obfuscated exe dropper works on my Fedora. Good to go forwards..

"Tutorial 8: Basic shading" gives exactly what it says in the title.. ambient, diffuse, and specular a la Phong. No news here, and nothing different from what I've been doing in WebGL before. So… this was a fast read-through. I'll be happy to move on to the "Intermediate" tutorials for the purposes of this learning project.

Next up: "Tutorial 9: VBO Indexing". Ahaa.. it was a short one. Basically, the learnings are that indexing happens using an elemant array buffer, and each shared vertex must have exatly the same attributes. So.. for cubes, prisms, and other sharp-edged objects, vertices must be duplicated anyway, because normals must be in different directions. For fun and learning, I could rebuild my cube using element buffers. Need to have each 8 corners facing 3 different ways, so need 24 vertices. Still less than the 6 * 2 * 3 == 36 vertices needed without indexing. Basically should use indexing, from that point of view.. And duplicate only as needed. In smooth shapes, no duplication is necessary..

Ok. Tutorial 9 is done.. I made another kind of cube that is drawn using an indexed element array and uses strided storage for vertex data (something that opengl-tutorial.com hasn't covered which I think is a bit of a minus for its use even for beginners..). I think I got more practice and learned something too, gradually. The code will be thrown away, but the next one will be that much cleaner. That's the thing about learning.

I haven't been tracking the size of my executable lately.. it is getting larger of course, because I'm now focusing on learning actual graphics skills and not using any time in micro-optimizing and zany tricks. It is pleasant to find this much capacity to focus to the task at hand every now and then.

On to the next tutorial, "Tutorial 10: Transparency".

2021-02-09T01:00 (2 hours 39 total)

Ok.. so the message about transparency was that it is hard. Read those research papers and keep it to a minimum in practice… But I played around with it a little bit. Now a transparent spinning cube. Nice. I'm planning to use some transparency in my production here, so it's nice to warm up a little bit.

To the next tutorial, "Tutorial 11: 2D text". I've been looking forward to this one, because definitely I'd like to have some blocky bitmap fonts in my visuals.. Also, some on-screen diagnostic information like FPS would probably be helpful always.

Ok, the story was to just build quads with UVs in a 2D font texture. I can do that if needed, but it wasn't completely what I had in mind for my production..

Not sticking around here.. moving on to “Tutorial 12: OpenGL Extensions”. Aha.. not much news in this one except that 3.3+ is already capable of most stuff. I won’t be needing extensions in this project. And I’m not planning to use older OpenGL where I’d have to query for extensions. Simple as that, I’m moving forward. Handling extensions is crucial in real software, so this piece is a must for a student to do on a university course. No skipping there.

“Tutorial 13: Normal Mapping”, a technique that I haven’t ever implemented, and actually a reason why I chose this particular tutorial.. I hope this is a good incentive to finally try this out. But later… it looks like I’ll have to sleep and work a bit again at this point.

2021-02-09T02:37 (2 hours 41 total)

Read and understood the basic idea and encoding of normal texture. So far I’ve been wondering if the “tangent space” is something very mystic, but it seems to be just a local frame that is aligned with the plane at a point on the said plane. Not complicated, but I suppose there will be quite a bit of code and data involved in maintaining the frames per vertex and ultimately per pixel. Probably a bit much overhead to be used in 4k intros, unless as a “main actor” / “primary effect”, but I’ll use this project as my incentive to learn some new things that I won’t be using here.. It is supposed to be about learning, not only producing.

I’m gonna need to draw some pictures about how the TBN matrix is computed, in order to understand it.. do it soon.. I’ll need a longer thought and understanding also about why tangents and bitangents are not to be normalized.. oh, but will they be finally, though.. before shading? they must be.. so the intermediate trick is just to manage triangle size, as the tutorial says.. Yep, the vertex shader seems to do the normalization in the tutorial example. It also does the inverse (by transpose) to move light directions to tangent space from cameraspace.

Ok, then after tangents and bitangents are figured out, they become just vertex attributes like everything else. Also the normal map as just another texture. And with the directions moved to tangent space, normal in that space is exactly the texture value (only decoded from [0,1] range by 2*tex-1).

Btw, observe that the first listing of computeTangentBasis() in the tutorial is not the whole story yet. It continues in the “Going further” part down the page.

Oh, I really want to get this implemented in practice! When do I have the time..

Right now I don’t have enough time. I need to get to sleep earlier than usual - it’s the orientation course of this spring opening tomorrow and I need to be running on full capacity or preferrably more.

But I have a few minutes. I’ll take a sneak preview of “Tutorial 14: Render To Texture”. That’s one technique that I plan on actually using in my production. I want to work with rendering in 1980’s lofi resolution with pixelated wireframes and then use the result as a texture in some fancier 2020’s scene. Sort of “history being played back in today’s boombox” kinda thing.. VHS digitized and shown in FullHD or that kinda thing.. I may want to do many of these, in fact.. as plenty as ultimately fits in the 4k bytes of obfuscated Forth.

As for implementation, I'll try to come back to normal mapping and tutorial 13 as soon as possible.

Short sneak previews of the rest of the tutorials, too. I count here 3 hours of just reading during separate time slots in the late afternoon. Sounds about right.

2021-02-09T23:48 (3 hours 44 total)

Revise and commit last edits; just think about what to embark on, next.

2021-02-11T20:39 (0.5 hours 44.5 total)

Continue the detour.. Towards some voluntary OpenGL exercises that probably won't be much related to the final outcome of this project.

2021-02-14T11:12 (1 hour 45.5 total)

2021-02-14T19:10 (0.5 hour 46 total)

Surprising problems with shaders, unrelated to Forth, which is not really surprising.. Black screen and crashing and the usual pain… Finally I think I have the tangent space conversion matrix figured out, maybe.. so, this little detour just needs a little bit of light calculations and a generated normal map for playing around a bit. I opt to do curved surfaces in some later project. I'll be happy to see some simple normal mapping on a non-curved cube surface.

I hope I can pick up some speed, because I need to learn the draw-to-texture things for this project.. oh, but I will learn that by doing the shadow mapping, and I could also do a nice normal map generator as a side-effect. It looks like I'll be learning a lot of OpenGL techniques in this part of my voyage. Necessary side-step. Will there be time to do an actual production? I can spare yet another hundred hours after this project if need be.. and another, and another… So. As of now.. let me just download all these new skills without thinking about deadlines. Learning is fun.

2021-02-16T01:18 (3 hours 49 total)

Not logging time for this project - I'm doing research and listening to inspirational atmospheres. One eye on the LaTeX, another one on the visuals of Tangerine Dream at Coventry Cathedral (https://www.youtube.com/watch?v=MHpiCCYcKOQ). Not actually sure if the visuals are original from the 1970s TV, but the overall vibe seems correct to me. It is 1975 but close enough to 1980's to be food for my entry.. exactly the kind of effects, glitches, pixels, and stuff I'd like to allude to. And, well, the soundscapes… would require more of a synth / sfx engine than I can afford in 4k, though.. but yes, food for the stylistic allusions, yes. Overlaid videos.. Oh, those guys were young in 1975! Now back to LaTeX though, both eyes. Focus..

Also, the Jarre music videos from the 1980s show directions for me here. But they're a bit too cheerful, perhaps.. I like the gloominess of these Tangerine Dream visuals.

Damn… that YouTube video totally stole my whole idea of a soundwave & scope visual embedded in a pixel starfield… F**, I'm so gonna steal it back! But with even fewer pixels and fewer polygons and fewer frames per second!

Ok, more songs for inpiration, a bit more modern but in the retro kinda cyber/steampunk way.. https://www.youtube.com/watch?v=TVaYeXGcA4E Good soundtrack for writing some science.

About the "modern take": https://www.youtube.com/watch?v=cdFHE73aOMI More: https://www.youtube.com/watch?v=2_kdXiP3jQ8 Lasers! Lasers and smoke, how have I missed those. Stacked video screens, monitor walls. And the overlays. In the 80s also those split views in TV and movies, showing action in different venues at the same time.

2021-02-16T13:08 (0 hours 49 total)

Oh, what a fight and frustration it was before realizing that I didn't convert my S" strings" to C-strings before calling GL.. Yep, Forth is impossible without factoring, factoring, factoring, all the time. Otherwise it's very difficult to spot a missing word. That's what Brodie said in Thinking Forth, yes.

Reminders on top of tutorials: https://www.khronos.org/opengl/wiki/Common_Mistakes (wouldn't have helped this debug round because I sucked at Forth today, not OpenGL, but something to really grasp later on!)

2021-02-16T20:01 (2 hours 51 total)

Tried for too long. Not possible to focus anymore. These hours were not very useful. Sleep.

2021-02-17T00:26 (3 hours 54 total)

Much better after sleeping, yes. Must remember to sleep before things become into a stall again. There are still issues with my normal mapping code. I think I got the idea correct, but not yet the implementation. I'll need more practice with everything. It has been too much time with this already. I'm past 2 ECTS studies here, anyway, looking at the hours. Must press on.

To the next few exercises: Render to framebuffer, and then briefly shadow mapping.

2021-02-17T22:54 (3 hours 57 total)

Off to work after yet another nice morning hour of hobby programming. This one is a good habit!

2021-02-18T09:01 (1.5 hours 58.5 total)

Some evening time, too. Almost finished with “Tutorial 14: Render To Texture”. Sooo much boilerplate code seems to be needed in setting up the framebuffer… I fear its gonna be a restrictive blob in the end. But I really want to do this framebuffer stuff in my art to get PIXELS. Maybe if I can then super overuse the framebuffer idea for all kinds of effects… probably so, because all that overlaid stuff and post-processing glitch effects are the goal .. Maybe a simpler synth / song then.. to fit everything in 4k. Or maybe I pick some 8k competition for the first public run?

2021-02-18T23:23 (1.5 hours 60 total)

Some lunch-hour coding and thinking, not much doing during the day.

2021-02-18T22:40 (1 hour 61 total)

OK, after a lot of the usual debugging pain, I’m now rendering to an off-screen texture and doing a post-processing pass using the framebuffer as an input texture. Really happy. There was even some time to play with the latest toy.

Regarding the debugging pain.. much of it would probably go away if I made a more proper design of the graphics lexicon. I’m gonna want a manageable matrix stack with easy vocabulary very, very soon..

But now, pressing onward - there are some more interesting and useful bits in Tutorial 14 of opengl-tutorial.org.. the reading of Z-buffer, multisampling.. some exercises. I decide to play just a bit more with these.. The Z-buffer thing will surely come around in Tutorial 16 which I intend to do.

Tutorial 14 is done, with exercises, so moving on..

2021-02-21T01:44 (3.5 hours 64.5 total)

I’ll just quickly skim through “Tutorial 15: Lightmaps” because I have no interest to use or create lightmaps for the current purposes. They are nice and should be used for actual work, of course, so the tutorial is nice to know, and necessary for any student of mine… Oh, that tutorial is a video.. I’ll have to tune in to it later, when I have access to headphones… So, I’ll move straight to “Tutorial 16: Shadow mapping”. First, just read it. This will probably take a while to implement, and I think I may have to do some refactoring first in order to not die of debugging frustration while hacking it.

2021-02-21T02:15 (0.5 hours 65 total)

Just reading the tutorial text and example codes.

2021-02-21T12:58 (3 hours 68 total)

Speed coding challenge of today: How fast can I implement a workable 4x4 matrix stack in Forth. I’m certainly going to need one in order to stay in control of transformations and scene building for anything beyond a single cube…

21:15 Ready, 21:33 Set, 21:41 Code! … Finished at 00:35, so three hours. A little bit disappointingly long. I was thinking it would be minutes instead of hours… Oh, well.. I failed in the step 1 of turning off the TV, watched a bit of Indiana Jones and the evening news. I couldn't resist to refactor the old stuff here and there to make it a little bit easier to continue from here. Transformations are now nicely under control, so it is possible to build meaningful scenes. I'm almost emulating the old OpenGL matrix… actually, I think I should completely emulate it, with its user-controlled push/pop! It's simple and intuitive… I prepared for using inverse matrices by allocating workspace, but I didn't implement them just yet.

Then, to finish off this weekend, an hour of playing with the toy, once again. The toy is now making sounds and visuals, so I've come a long way from playing with the ." Hello world" stuff earlier in this project. So I've seen places and come a long way.

Next up: build a test scene for trying out the shadow mapping tutorial stuff. Maybe think about light and camera positioning while at it.

But, alas, of course, a week at work.. I hope I can do some morning and lunch break coding. Weekday evenings are a bit too tired to get things moving properly.

2021-02-21T01:18 (4 hours 72 total)

Today in short: Completed yesterday's ideas 1h, reading Internet fora about VAOs and stuff 1h, revising matrix stack and implementing inverse matrices, 2h. Total of 4 hours - quite a good pace on top of a complete working day. Morning, lunch break, and evening well utilized. Didn't read any news nor watch TV. Momentarily well done in life, actually.

2021-02-21T23:18 (4 hours 76 total)

Reading about matrix inversion, 1h, trying to size-optimize a 3x3 inverse, 1h. Facts: Direct inversion and one call to it adds 150 bytes (from 5668 to 5817 at the moment) with my best attempt at compact code. The continuous update of inverse along with forward transformation is approximately 120 bytes (from 5548 to 5668) with no effort in size optimization. And it is 4x4 inverse, not only 3x3. There seems to be a winner algorithm, just in case inverses are needed.

Wonderful night-time studying and morning coding. Now I must run to work (the whole 2 meters run from the kitchen table to the home office…)

2021-02-22T09:15 (3 hours 79 total)

Lunch hour reading of Internet fora about VAOs and stuff. Also about some offerings of OpenGL 4+, but I'm not going there yet. OpenGL 3.3 Core is good for this particular project, at least as of now.

Back to the shadow mapping tutorial now.

2021-02-23T16:01 (1 hour 80 total)

A test scene with scaled boxes finished. It's nice to see the results of playing with the toy, but the toy is quite hard to drive at the moment. There is a lot of redesign ahead before it becomes pleasant to set up scenes. A simple syntax for float input would help.

Next up: Need some boilerplate to make the shadows.. Orthographic projection and the "look at" matrix. Will be useful for many things, so just go and do it.

2021-02-23T18:30 (1 hour 81 total)

Why is it never any easier.. to spot that missing minus sign? Again, I had to spend time on a problem that had nothing to do with Forth, or anything else that is any new to me at this point of life: 2 hours with the good old wrong sign problem. Maybe next time it will take a few minutes less to spot these.. In theory, I know to look for these in equations. And I did. So, why did it take so long to *see* it? Maybe the problem is indeed that it was a missing sign, not a surplus one… even harder to spot that something does not appear than it is to spot that something strange appears (thinking of the Monkey Business https://www.youtube.com/watch?v=IGQmdoK_ZfY once again; watch that video if you haven't already, btw. Cool stuff.)

Ortographic projection is in place.. I'll do the lookat too because it will be universally useful. But maybe later. Sleep now. Must remember to.

2021-02-23T23:32 (2 hours 83 total)

After sleeping, I realize that I don't need the lookat for the current purpose. So, postpone to even later time. I can do the shadow exercise very well with the current transforms.

Meanwhile, add to aesthetic influences Blind Ltd. approach to Solo: https://www.starwars.com/news/blind-ltd-solo

Nice morning hour of coding, with progress and nice refactorings.

2021-02-24T08:55 (1 hour 84 total)

+1.5h micro-optimizations. Stop this madness! I specifically told myself not to do this now. Only creative work!! Slipped once, don't slip twice. Ok, admittedly it was partly refactoring that will make creative work easier and faster towards the end..

Back to the tutorial 16, shadow mapping.

2021-02-24T22:00 (3 hours 87 total)

I talked to our students about this project for a couple of hours. That was working time for which they give me the monthly salary, ultimately from taxpayers, so I refuse to count those couple of hours in this hobby project. Make a note to self that a colleague pointed out some possible next steps in thinking Forth, because I mentioned that I probably want to be thinking Lisp after this current project: http://soton.mpeforth.com/flag/jfar/vol4/no4/article6.pdf

Students seemed interested.. I can't wait to get a production out so I can properly publish this whole project..

2021-02-26T13:30 (0 hours 87 total)

They are showing Blade Runner on TV. How did I forget Blade Runner (1982). That masterpiece should be one of the top influences in the kind of aesthetics I want to allude to. All those “high-tech CRT displays” and neon lights of the futuristic year of 2019. Deckard zooms into the grid of squares and so on… Blinking lights, epilepsy warning…

+1h more debugging. Veery silly error! Veery silly, and again nothing to do with Forth. Raw OpenGL is sticky business with the state. I put a glDisable(GL_DEPTH_TEST) for the purpose of debugging and *helping* me to understand a problem and ooph.. totally forgot to force glEnable(GL_DEPTH_TEST) in the actual draw. No wonder I get a Z buffer of all white. Conclusion: I should have a lexicon that hides all these little things so that such surprises would be harder to create in the application code. That won’t be an easy thing, I believe.

2021-02-27T15:42 (2 hours 89 total)

Painful debugging of some yet unknown misunderstanding.

Code is easy, understanding tough. Face and palm connect. I was doing the light transform in the camera space while it was supposed to be done in world space. So there was double trouble from a double transform. Fixed now with the face so deep in the palm it can’t get deeper. 2 hours for this bruise of the forehead.

Time and pain aside, I’m happy. Shadows are in the right ballpark, but they have both acne and peter panning. This was expected. I’m happy to understand the basic idea, and now I know at least a few caveats that a beginner is quite likely to run into. Good.

2021-02-28T02:12 (2 hours 91 total)

Time and pain, time and pain. Debug my inverse matrices, find out a math thinko, so inverse matrices could be OK after all. But *unnecessary* because GLSL computes matrix inverses since version 1.40. I don’t think it’s gonna be useful to invert much anything on the app-side for a 4k intro. Well, it was fun and educational in addition to being painful and useless for the task at hand. A definition of learning?

2021-03-01T01:38 (2 hours 93 total)

And pain, and time, and pen and paper. I wish my younger students do this kind of exercises properly while they are young. It gets harder when you get older. So, what about my older students? No problems, but they’ll have to do this kind of things the same way that I do: old and hard. In the end, it is also rewarding and fun. The fun is a little bit hard to see when you’re just feeling dumb with the the fleeting logic of 3D transformations and their equations… those moments of trying to understand a thing for the first time, before getting to the point of satisfaction when you understand something in order to simplify it (and feel stupid once more for not being able to simplify it many hours ago…). I guess the fun comes later, when looking back at the learnings that have become clear since the beginning. Until they get forgotten again, that is. I think the satisfaction somehow gets even bigger when I'm older. When younger, the new learnings didn't really connect to a bigger picture. I think I like this mature learning even more than I used to like learning while younger. Even if it takes more time and effort now.

2021-03-02T01:06 (2 hours 95 total)

Done with tutorial 16. Whoa. Just like it says in the beginning: easy to understand and implement the basic idea of shadow mapping. And very, very hard to get it look right and beautiful. I think everyone should have their own first battle with shadow mapping early on in learning real time rendering. And the first battle with normal mapping and these other older tricks in the books. In that sense, opengl-tutorial.org has good choices of content and order of presentation.

For completeness, I'll read through the final tutorials 17 and 18. I won't pick up any code for those at the moment. I'll be busy enough with finalizing this project that… ehm… did originally have some overarching goal that I've forgotten about with all this new and exciting learning…

And now, a note-to-self of grave importance:

- Full feature freeze now, and focus on producing something that I would dare to enter to a competition as "The Old Dude" in the next Instanssi, after this COVID-19 lifts and we can hope to have one. Or in some other friendly neighborhood party that welcomes beginner quality entries and "filler acts"… It can even be in a full demo compo if it doesn't fit in 4k with the current obfuscation, if need be. And something that yields a making-of presentation as an Instanssi "Summamutikka" entry, too. That was one of the original goals from day one, after all.

2021-03-02T18:07 (2 hours 97 total)

Ok, so "Tutorial 17: Rotations" of opengl-tutorial.org is a very brief survival guide about using quaternions with all theory deferred to auxiliary readings. As such, it is not very suitable for advanced university studying, but at the same time it is an obligatory "short" to the toolbox of any intermediate level graphics coder, so if I was to make my students do the particular tutorial, its part 17 would be mandatory for full credit, I guess.

I'll check 18 : Billboards and particles. Billboards is a five minute read and minutes to hours practical coding, depending on the prerequisite skills gained before the attempt. So, it goes to the list of mandatory things to visit on a course.. Then the bit about particles? Ok, it's about particles and instancing. A good primer to get into exploding particle animations. We happen to have a separate course on the whole topic of physics models, so particles would be on the other side of a course boundary, both for the student and for me in this project. I'm not planning to do particles for this one.. well, never say never, … I know where to look for a first hands-on tutorial now.

At some point, I want to check out de Vries' tutorial too, but it must be later. No more time to spare with tutorials now, I must get to the artsy fartsy side of creation right about now.

[Note to self: Language has been checked up to this point already; I don't think I can make it any more proper English with my current skills.]

2021-03-02T19:30 (1 hour 98 total)

Situation check:

I started searching for OpenGL tutorials at 12 project hours and finished one of them at 100 hours. That is 88 hours of OpenGL learning. That is a tiny bit more than 3 ECTS credits which would have been 27 x 3 == 81 hours. So, I can pick up the stuff in 3 credits using a language I've not yet programmed much on, on my own implementation of the language that is pretty suboptimal in debugging capacity. With zero possibility of copy-pasting working code from the tutorial without careful porting to Forth. Meanwhile, I built a lot of the necessary boilerplate that is available free-of-charge in any actual matrix library. Can I expect a master level student to do this in 5 ECTS using C++ after they already know the language and matrix math? Ya bet I can. And surely I can expect even more of them, but implementing their own interactive OpenGL 3.3 engine from scratch using a tutorial as reference is a safe minimal requirement for passing 5 ECTS. This is good to know. I feel it in my guts that this is right.

As for myself, I have pretty exactly 1 ECTS worth of time to get my creative practical work done. Can't expect it to be spectacular. Must focus.

[Accounting for bad hour counting on 2021-02-05, +2 hours to the next one]

2021-03-04T01:00 (5 hours 103 total)

At this point, I need to wrap up the current proof-of-concept, start anew, and start killing darlings. Copy-paste-modify with focus on the target and nothing but the target.

A little bit less than an hour first to make the POC code show the incremental progress starting from tutorial part 9, which was the indexed cube with texture.

Next up:

- Tag this point that is the final "proof-of-concept"; after this there will be only final and publishable productions.
- Make a fresh start and a new source file for the first production. Start from scratch and copy-paste-modify usable parts.
- Redesign the lexicons.
- Kill the many darlings that won't be used.
- Prepare "assets" and ideas re-useable in the final showreel.
- Remember to look at page 1 of this episode: Don't go optimizing anything.

2021-03-04T11:52 (1 hour 104 total)

Good to go for the tag.

Back to business, after the detour of learning

2021-03-04T21:30 (1 hour 105 total)

Tag made, and now quickly re-working everything to be leaner, meaner, simpler, easier.

I'll also benchmark the sizes of program functions more seriously now. Purpose of this benchmarking would be to prioritize future (micro-)optimization efforts.

Current overhead from bootstrap code only, i.e., current minimum boilerplate (size in bytes + one space + filename):

793 prod1.runner

I know I can shave some bytes away from this part, but I'll have to understand that it will affect less than 1 out of the 4 kilobytes I'm aiming at. Not a lot of wins possible for the hours spent in working on it. Yet, every 41 bytes constitutes as much as 1% of the 4096 that is the target. So every hour that gives a 20 bytes saving is an hour to celebrate. This is a hobby, we have to remember…

Anyways, this is a good initial baseline measure to understand: There is now some 793 bytes boilerplate, most of which can be considered overhead from booting the Forth interpreter. This measure should be compared to the very smallest possible ELF binary. There are a lot of bytes lost in this battle, right? So, they should be somehow regained in further steps in which Forth would do things more efficiently than competitors?

I'll go from here. There will be so much more from linking and calling the necessary functions of SDL and OpenGL.

And yes, the build script and file structure look much nicer and easier to understand after these couple of hours of thinking about them. It's funny and embarrassing that they're still so messy. Yet, a step forward is not a step backward.

2021-03-05T02:33 (3.5 hours 108 total)

2021-03-05T16:40 (0.5 hours 108.5 total)

One eye on the TV and the other one as useless. I just punish myself by counting these hours of my usual hell on earth. I wanted to do something and then I stared into a media. It was Blade Runner 2049, but I don't like the movie and I know it. I don't want to see it. Saw. Pain. Punishment. Thanks. Sleep.

2021-03-06T02:00 (1.5 hours 110 total)

Size of empty window and silence:

1971 prod1.runner

This includes all the boilerplate for making platform calls. Yet, there must be room for optimization especially in this part. Next increment to a beeping cube perhaps?

2021-03-07T15:45 (2 hours 112 total)

Oops.. I went into the optimization trap. Focus. Snap out of it..

2021-03-08T19:05 (2 hours 114 total)

And it became a debug trap. Dark side. Stay away; focus.

2021-03-08T19:05 (2 hours 116 total)

Size of empty window and the dull proof-of-concept audio track:

2431 prod1.runner

Size of four circling squares and the audio:

3840 prod1.runner

That could be a little bit disencouraging.. But it is also a good benchmark. I still think I can get this beast minified. The current code has a lot of extra weight everywhere. With this 1.4 kilobytes came the whole matrix stack thing (with inverse matrices that aren't needed) and a lot of boilerplate that can be re-used for other things. Also, absolutely none of the current dog vomits are optimized for size. (The earlier 4-hour detour was for benchmarking convenience and not the real thing… why did I do that, why…)

To look at it from another soothing perspective: If I manage to not do any optimizations now, like my PLAN was in the first place, then I only have 240 bytes to fill with some not-too-dull audiovisuals, and I will have reached the main goal of this project episode: I'll have a 4k production, even if it is not yet the real masterpiece but a small step on the way. But I'll have to focus, because I only have 8 hours do do that before wrapping up this episode. So… Focus, damnit! And now sleep and work a bit.

2021-03-09T02:07 (6 hours 122 total)

From here: Quickly rid of inverses! Take the framebuffer thing and then just fake it or make it but nonetheless wrap it up.

Quickly rid of the inverses:

3724 prod1.runner

2021-03-09T17:38 (1 hour 123 total)

Next, quickly take the framebuffer thing.. But first, maybe quickly a more convenient shader and uniform variable abstraction..

Ok, I quickly revised the shader and uniform creation and uniform variable storage:

3705 prod1.runner

It looks like not much was saved for an hours work, but in reality the change enables mass production of shaders with practically unlimited number of uniforms for a 4k production. And much cleaner application code.

2021-03-09T20:46 (1 hour 124 total)

Array buffer now behind a creating word interface.. Lost a battle here:

3738 prod1.runner

Hunch is that this was to win the war. Application code is simpler again, and library is more structured. Nice morning coding again. Now run two meters to work again.

I'm going to need textures and framebuffers in the end. There is no time to put it all in place until my allotted 4-5 hours is at end. If I want a production like the plan was. I should remember my own "Feature freeze" from before and just make more squares as they are. Continue in the next episode. Focus!

2021-03-10T09:25 (2 hours 126 total)

Read more about OpenGL. Should redo a lot from scratch, including ways of thinking. Should postpone doing that to the next episode, really. Just do something now with the current code. Focus!

2021-03-10T21:41 (1 hour 127 total)

A smiley texture once again. Nice visual but whoa it becomes big in size:

3974 prod1.runner

Optimistically, none of this is optimized yet. But still, how can it be 230 bytes for just adding a texture. Ok, ok, again it was adding code that can mass-produce all the kinds of textures I'll need, but still it is 5% of the whole byte budget in a 4k executable. Will be interesting times to look into these, and optimize…

Oh… and I'm really running out of time here! I'm still under 4k. Can I do something sensible just with a textured square.. in the remainint 1-2 hours before I'd like to stop for intermediate self-evaluation…

2021-03-13T01:55 (2 hours 129 total)

I just want to know how much it now costs to have a second bitmap image as another kind of "sprite" to render… Just do it and see…

The quite suboptimal data only seems to be 30 bytes:

4006 prod1.runner

Good. That can be optimized or actually dropped altogether, because it is basically plain stupid at the moment.

Actually drawing either of two textures:

4026 prod1.runner

That means 20 bytes for the code that changes the texture while rendering. Total 50 bytes going from having one texture to two, interchangeable textures. I'm not making any conclusions yet. I'm just fathoming the current space of possibilities.

I'm really running out of time here. I can totally relate to the feeling my students must have when they should be finishing something in 135 hours (the 5 ECTS limit) and they're not just there yet and want to spend all their life to get the exercise answer right in order to score the highest grade of 5/5. (At least I hope they aim to that side of the spectrum instead of the other one…) It is almost impossible to let go when you're slowly progressing but the time is basically up. How to accept the lower grade and move on in life? Oh, what a difficult and important thing to learn in addition to the subject matter!

I'm gonna be exemplary here, and accept a low grade after the time is used, even if the product is unfinished. In today's world, we cannot afford to exceed the time limits of pretty much anything… In my opinion, the time limits are unreasonable for proper learning, and the situation is very unfortunate. I would fight every battle I could to change this back to what it used to be in my own degree study times. But, unfortunately, we are in a situation where time is money and both are ever more scarse. Vote better next time, fellow citizens of the Finnish democracy… no more cuts to education, and at least some of the old resources should come back like right about now.

Nowadays, we have to stick to that maximum of 135 hours per 5 credits, and get our grade based on the result. Otherwise we starve. Can't do the 160-180 hours anymore to score grade 5/5 for every course. Those days are OVER. I don't know where the finger should point exactly, but some of the people and parties in budget power during the last 10-15 years or so have done a very good job in destroying possibilities of quality study. Sigh. Drop a tear. Back to business. Today, we are robots that can still self-assess at each 135 hours. Turning to that now..

2021-03-13T03:07 (1.5 hours 130.5 total)

Some ad hoc mathematical beat pattern to the music… interesting…

Beginning of the end of a cycle

2021-03-13T03:50 (2 hours 131 total)

Already thinking about the self-assessment template.

2021-03-13T21:20 (1 hour 132 total)

I want to make the time-quality compromise explicit in grading. If any section would be 0/5, the whole course remains unfinished in the failing state. At least 1/5 of each category must be achieved for credit.

1. Time used for this 5 ECTS learning episode.

   Must be tracked and documented from the very beginning to the very end, at least on the detail level of “what I worked on, when, for how long at a time”. Tracking for example “thinking time away from keyboard” is very fine, when you have thought about the issues of this course. Use common sense. Find a convenient granularity for yourself. I have tried to provide one example in my demoscene hobby projects. But make it in your own way, as long as the above minimum requirement is met, and a total hour count can be observed.

   0==FAIL: less than 70 hours (reeks of no interest to the topic or the whole field of study! Stop this madness, do yourself a favor, and find something more interesting to do with your precious one life than this kind of topics!) OR no documentation provided (won’t fly; redo from start, with time tracking!)

   1/5: 70-81 hours (about 50% of what was actually expected. Absolute minimum! Rethink motivations!) OR more than 168 hours (one week or more of extra work. STOP. Start next course instead!) 2/5: 82-94 hours OR 160-167 (four full working days, almost a week. Hands off already!) 3/5: 95-108 hours OR 152-159 (three full working days too much, almost time for a 1 cr course) 4/5: 109-121 hours OR 144-151 (two full working days extra time is a bit much “finalization”) 5/5: 122-143 hours (target is 135; one full working day extra finalization is appreciable.)

2. Ability to find focused time, “getting into and staying in the Flow”

   0==FAIL: (impossible to fail because of this category)

   1/5: I basically failed in allocating focused studying time - I had lots of distractions all the time, and I had very short periods of staying on topic and actually working towards the course goals. 2/5: Between 1 and 3 3/5: It feels as if 50% of the hours were well spent and focused on the task at hand 4/5: Between 3 and 5 5/5: Almost all, say 95% by gut feeling, of the hours were spent without distractions, moving towards the goal.

3. Technical requirement achievement

   Must be verified by the actual code in question that is shown to the teacher along with this self-assessment.

   0==FAIL: I was unable to produce working code for 50% or more of the required tasks. 1/5: 50% of required tasks produce expected results for clean input with my own code 2/5: 70% of required tasks produce expected results for clean input with my own code 3/5: 80% of required tasks produce expected results for clean input with my own code 4/5: 90%

of required tasks produce expected results for clean input with my own code 5/5: 100% of required tasks produce expected results for clean input with my own code

4. Technical quality achievement

   0==FAIL: (impossible to fail this category)

   1/5: Poor quality: My code is not commented, and it would be difficult for other people to understand. 2/5: Between 1 and 3 3/5: Good quality: Another developer could pick up my code and both use it for their own applications or develop it further. My comments explain all of the API and the reasoning behind crucial choices that differ from those expected by an application programmer. 4/5: Between 3 and 5 5/5: Brilliant: I am proud of my code. I feel confident that I could continue its development easily and that it could be used easily by an application programmer by reading the documentation / examples that I have included.

5. Learning achievement

   Must be explained in words: How and why did you learn or not learn each of the topics that were part of the course curriculum? (The specific list of curricular topics could be copied here for convenience on an actual course)

   0==FAIL: (impossible to fail this category)

   1/5: Several concepts remain vague: I find it hard or impossible to give examples of what I learned of the intended subject matter. 2/5: Between 1 and 3: I was able to differentiate between topics learned and not learned. I gained skills in some of them. 3/5: I learned a lot: I know what each topic of this course means in the big picture. I know some of the topics so well that I could use them at work or teach them to others. 4/5: Between 3 and 5 5/5: Brilliant: I left no stone unturned. I master every detail on the level required in the target skill description.

6. Learning extent and zone of development

   Grades 2-5 must be explained in words: Which were the specific things new to you.

   0==FAIL: (Impossible to fail this category))

   1/5: I learned nothing new; I only applied things I already could do 2/5: Between 1 and 3 3/5: I learned a lot of new things (which can be mandatory elements, if those were new to you, or voluntary challenges like a new programming language, library, syntax, design pattern, development tool, etc.) but mostly I stayed in my comfort zone with familiar tools and methods. 4/5: Between 3 and 5 5/5: I made sure that I don't do things "the old way" but try to be working with something new to me all the time (95% by gut feeling).

7. Look back and forth

Obligatory part, but not graded

0==FAIL: Missing

PASS: Each of the following questions are answered.

7.1 Reflect back: What was the easiest thing for you to learn or do?

7.2 Reflect back: What was the hardest thing for you to learn or do?

7.3 Reflect back: What was the most useful thing that you learned?

7.4 Reflect forward: Where do you plan to use the things you learned here?

7.5 Reflect forward: What would you like to learn next about these things?

7.6 Reflect forward: How and when do you plan to do that learning?

7.7 Reflect forward: What will you try to do better in your future studies for things to go smoother?

8. Overall grade and plea for adjustment

   Basis of your grade is the arithmetic mean of the six graded sub-areas above. Round to the nearest integral number.

   If any of the sub-grades are 0, or any required verbal explanations of the self-assessment template are missing, you will have to do more work to complete the course.

   Compute the mean here: (g1 + g2 + g3 + g4 + g5 + g6) / 6 = ____________ ~ ______ (rounded)

   Finally, if you feel that the mean is not a justified grade for you (in either direction), you may plea for adjustment of +1 or -1, with the reasons you feel justified.

   Optional plea: __________________________

   The teacher reserves all rights to either accept or decline your adjustment plea and to decide your final grade in the whole 0-5 spectrum, adhering to principles of equality and other relevant laws in effect in the EU and Finland. If the grade differs from your self-assessment, the teacher will give justifications to the decision.

   Remember that dishonesty in self-assessment is a serious offense, and will lead to investigation and the most severe punishments possible if and when detected! It is very important that the claims you made about your exercise achievements can be verified also in the source codes produced as part of your course work. For example, a copy-pasted tutorial code without your own comments hardly warrants you to say you have learned a thing. The situation immediately changes for the better if you have tried to build the code from scratch first, using reference manuals and only the narrative of

> the tutorials, and then describe a controlled path of discovery from your own attempts to the fully-working examples available in tutorials.

There. Personally, I'd like to be evaluated pretty much like that in a self-study programming course. I'll do that to myself here to try it out.

And, by the way, look what I just did: I gave a full working day of extra finalization time that doesn't yet lower the grading. After careful thought - it is a must! And now… I'm gonna actually use that 8 hours to feel better about the other parts!! When talking about an endeavor of 135 hours, which is like 17 days (long ones.. 8 hours a day), it would be very awkward to say that you can't go just a little bit below or over that 135.00 hours.. 8 hours slack either way sounds very well justified. I'm actually quite excited about these insights. Part of my "extra-curricular learnings" in this project, by the way..

So… 9 more hours suddenly appeared in the schedule. I try to use as little of it as possible, in post-deadline-panick-ey wrap-up.

2021-03-14T17:39 (2 hours 134 total)

Make an actual bitmap using bits.. Nah.. soo much space (some 200 bytes) needed for 25 bitmap fonts of 6x6 pixels.. Could be optimized perhaps, but not now. Drop the idea.

2021-03-14T20:00 (1 hour 135 total)

Bad news: The Summer is Coming

And then there was work, no time for hobbies. But I'm back after what seems to have been two weeks. I've forgotten most of this, but I think it is in fact good. I'll have to reflect on the necessity of a complete pause later.

Now I'm just trying to get this thing done before I start to lose credit, according to my new assessment template!! I did allow only a working day of slipping..

2021-04-03T02:00 (2 hours 137 total)

Wire-kinda done.. Next up: The framebuffer..

2021-04-03T03:00 (1 hour 138 total)

Started with the framebuffer.. some way to go still..

2021-04-03T17:19 (1 hour 139 total)

Now I get lofi wireframe cubes in 320x240 pix rendered in framebuffer and then brought to a texture. Nice. But I'm over hours, badly. Really, now just some cubes and call it a compo-filler! Also, it is still 4.4k…

But yeah, looking good and promising..

2021-04-04T03:30 (2 hours 141 total)

Cleaned up test2.fth, I hope for the last time.

2021-04-04T04:12 (0.5 hours 141.5 total)

Yep, made some nicer-looking wireframe cubes. Now I'll just simplify and reduce until it fits 4k, and call it a compo-filler and move on. And wow, I only have an hour before it starts to cost credit even according to my new permissive self-evaluation template…

2021-04-04T15:48 (0.5 hours 142 total)

The end is near.. 4.2k with Zopfli.. Half an hour left before negative credit. Break, jog, then full speed once more. Quick notes of sizes before deleting benchmark code:

4438 prod1.runner current, with cube and framebuffer

4350 prod1.runner many textures, square only, no framebuffer

4046 prod1.runner one texture, square only, no framebuffer

2591 prod1.runner black window with music

2125 prod1.runner black window with silence (SDL setup overhead)

1128 prod1.runner clean exit without setup (Forth bootstrap + residue by the current obfuscator)

So, here's the data for further analysis about where to focus first in size optimizations.. I'll get rid of the versions now because it's quite distracting to keep them up to date. I'll make new benchmarks later if need be. Better off without at the moment.

2021-04-04T16:50 (0.5 hours 142.5 total)

That's how I planned it. And then I bought a house. Almost no idea what happened in the meantime, but it took exactly 4 months to take place and none of it was with this project. Spring / early summer is not the time for hobby projects, it seems. Maybe without buying houses?

But at least I come back to this project with totally fresh eyes again. I need to read some of the above notes to have any idea about what was going on here.

2021-08-04T15:30 (0.5 hours 143 total)

Reading of codes for 1 hour. I think I found the places to modify in order to finally wrap up this part of the journey. After these long breaks, I'm always very happy to find my old comments that make it quite easy to bring everything back to the brains again.

2021-08-06T16:50 (1 hour 144 total)

A little bit more interesting geometry.. also just picking up the pieces after a long break.

2021-08-06T18:30 (1.5 hours 145.5 total)

Over all time budgets. But at least I'm back and progressing. Visuals OK for a few second party-filler. Now just melt away some 200 bytes.. Zopfli packst the current one in 4260.

2021-08-09T05:01 (3 hours 148.5 total)

What a wonderful day: Imagining I'm still on vacation - can do hobby programming with no interruptions. I can spare this afternoon. There will be just enough work during the winter, I believe.

2021-08-09T14:05 (1.5 hours 150 total)

The afternoon turns into an evening. Still with the project, trying to do what was supposed to be done in April. Better late than never. Nine bytes to shave, and it's a party-filler I could throw in at a friendly neighborhood compo.

2021-08-09T19:31 (5 hours 155 total)

Actual end of the cycle

Hooray!! I have a first production! It can be packed into 4095 bytes using Zopfli with 40000 iterations! (FSF's gzip -9 gives up at 4252 bytes). It volunteers to using desktop resolution, auto-stopping at maximum running time, hiding the mouse cursor, and removing dropped content from /tmp after exit. Most importantly: It is not an instant death by boredom. Instead, it is 44 seconds of lofi pixel mayhem that can be watched through once or twice, I believe. It can be a 4k intro compo entry in Instanssi 2022, if they are able and willing to arrange the wonderful party.

It took too long to wrap up, even in "project time". I need to learn how to cope with these real-world interruptions somehow.

2021-08-10T00:23 (1 hours 156 total)

Now.. Finally, I get to try out the self-assessment template I devised earlier (2021-03-14) . This will be interesting. And quick-ish, I hope.

1. Time used for this 5 ECTS learning episode.

   3/5: 95-108 hours OR 152-159 (three full working days too much, almost time for a 1 cr course)

2. Ability to find focused time, "getting into and staying in the Flow"

   3/5: It feels as if 50% of the hours were well spent and focused on the task at hand

3. Technical requirement achievement

   Must be verified by the actual code in question that is shown to the teacher along with this self-assessment.

5/5: 100% of required tasks produce expected results for clean input with my own code

4. Technical quality achievement

   5/5: Brilliant: I am proud of my code. I feel confident that I could continue its development easily and that it could be used by an application programmer by reading the documentation / examples that I have included.

   (Note: This is a tough one to assess in a project that aims to break all rules in order to minimize exectuable file size.. Nevertheless, I find myself proud of my code and the fact that I was able to pick it up quite easily after a total break of 4 months)

5. Learning achievement

   Must be explained in words: How and why did you learn or not learn each of the topics that were part of the course curriculum? (The specific list of curricular topics could be copied here for convenience on an actual course)

   5/5: "Brilliant…"

   So… to explain this in words… This episode was supposed to be a "practical" instead of about theory or skills: I had the single goal of creating a 4k intro that could be put to a compo. Did it. But it turned out that I needed to learn some new OpenGL, so I peeked into some tutorials and went through one myself… This lead me to think and learn a bit about how to arrange a self-study course on the topic. So… I learned and discovered a lot in theory and practice outside the stated syllabus of this practical episode. It is a natural byproduct of any practical, though, of course. You learn by doing, because doing interesting stuff requires learning first to enable the doing…

6. Learning extent and zone of development

   Grades 2-5 must be explained in words: Which were the specific things new to you.

   3/5: I learned a lot of new things (which can be mandatory elements, if those were new to you, or voluntary challenges like a new programmin language, library, syntax, design pattern, development tool, etc.) but mostly I stayed in my comfort zone with familiar tools and methods.

   OpenGL 3.3 was all new to me. I also took a peek to 4.6 and Vulkan while at it, so I got some nice glimpses of current topics all new to me. Yet, I have to say that I purposefully tried remain in old comfort zones especially with audio code which was a very quick recitation of old learnings. I think it was a necessary balancing act between all the goals in this episode: Get something finished, learn graphics, … so, focus on those two and do everything else by old habit for now.

7. Look back and forth

Obligatory part, but not graded

PASS: Each of the following questions are answered.

7.1 Reflect back: What was the easiest thing for you to learn or do?

It was surprisingly easy to grasp some graphics techniques that I thought to be harder.

7.2 Reflect back: What was the hardest thing for you to learn or do?

Once again, it was impossible to start wrapping things up early enough. I think I should have left some of the OpenGL tricks to some later project and just be done with the practical. Early stopping is a very hard thing for me. It feels like a real disability.

7.3 Reflect back: What was the most useful thing that you learned?

Rendering to off-screen framebuffers and using those as textures. Definitely a useful key skill in day-to-day graphics.

7.4 Reflect forward: Where do you plan to use the things you learned here?

Graphics skills everywhere in graphics. Forth skills… well, I never got into this project planning to actually use any of it. If somebody needs some help with Forth, I might know a thing or two about it by now.

7.5 Reflect forward: What would you like to learn next about these things?

My dive has just started. After implementing my own Forth as a 4k intro platform, I plan to move on to Lisp and then other languages. I realise that I need to recapitulate and learn more about quite fundamental computer science stuff. Forth was a nice stepping stone with its trivial parsing mechanism.

Regarding graphics, I need to learn even newer OpenGL (4.6+) and Vulkan, maybe Metal, and current things like GPU-assisted raytracing. Oh, and the deep network stuff on GPUs, too. So much to learn, so little time…

7.6 Reflect forward: How and when do you plan to do that learning?

With the little and ever-diminishing time remaining for learning.

7.7 Reflect forward: What will you try to do better in your future studies for things to go smoother?

Same question again - a question that must be asked. From me, the same answer: I try to stop the madness of reading all the news of the world and then the rest of the Internet. I try to break the spell and find more time with interesting stuff like learning new things.

And I need to balance my working hours to the 1612 hours required of me each year. Doing too much over there, which is away from nice hobbies like this.

8. Overall grade and plea for adjustment

   Basis of your grade is the arithmetic mean of the six graded sub-areas above. Round to the nearest integral number.

   If any of the sub-grades are 0, or any required verbal explanations of the self-assessment template are missing, you will have to do more work to complete the course.

   Compute the mean here: (3 + 3 + 5 + 5 + 5 + 3) / 6 = 4.0 ~ 4 (rounded)

   Finally, if you feel that the mean is not a justified grade for you (in either direction), you may plea for adjustment of +1 or -1, with the reasons you feel justified.

   Optional plea: _______ No plea would be needed in this case.. I think 4/5 is justified. It would have been 5/5 if only I could have cracked the mystery of the procrastination versus focused training nut. But it remains the number one disabling problem to be dealt with.

OK.. Quite nice. I can imagine some of my students not really appreciating the opportunity of reflection. I don't know if I enjoyed it too much myself. Yet, in so many ways, all this reflection is useful. As of this first try, I don't find many problems in the template. I think I could use it in a real course, why not.

It took 1.5 hours to fill in… but this happens in an already tired state of mind, and I did fix a lot of stuff in the template while doing the assessment itself. Totally doable by a student.

Phew.. Done with this episode! On to the next one!

2021-08-10T02:05 (1.5 hours 157.5 total)

Part 6 / Episode VI: The Practical, Part 2

(Learning diary of a hobby project.)

Transition between the previous episode and this one

This should be a quick one now. Most urgently I need to read about the authentication changes at github. Password authentication ends in 3 days..

2021-08-10T14:37 (0 hours 0 total)

Hmm.. I think I got it.. try to push this line using my newest access token instead of password. Seems to work fine. Also read lots of Github docummentation. Interesting stuff, but not related to this project, so I'm not starting the clocks yet.

2021-08-10T15:30 (0 hours 0 total)

Making some space

I need to make it pack better. I really believe it is possible, but I'll just have to find the right ways to make it happen. I'll attack the token stream format first, and try to get closer to one byte per word..

So, it's benchmarking again. I'll use prod1 as a measuring stick. FSF gzip. Starting point:

4247 prod1.runner 8386 prod1.obfuscated.bin

Latter is the binary payload. I intend to minimize both.

Started to look at the obfuscator again. The part that I'm not too proud of.. quite messy. I'll try to sort it out just a bit; otherwise it will be hard to tighten the obfuscated format.

The messier the code, the more time it takes to touch it. Should these hours have been spent in making it less messy in the first place? The perpetual battle.

2021-08-11T12:17 (4 hours 4 total)

Yep, still imagining I'm mostly on holidays. A couple of work meetings in between, though, but now back with hobby programming. Nice progress towards trying out the dim bulb idea of not using separating whitespace in the obfuscated format.

2021-08-11T16:31 (1 hour 5 total)

Wow, that was a difficult debugging nut to crack. Such a stupid mistake again. But it got fixed. I got rid of some of the whitespace in the format. New measures:

4118 prod1.runner 6899 prod1.obfuscated.bin

So some 1500 spaces were deleted from the data but only 130 bytes saved after packing. This was to be expected, because the spaces did pack well. 10 hours have been spent in gaining 130 bytes of creative space… Disappointing? Not really. I need all the space possible by any means, and the obfuscator code was actually improved quite a lot while attacking this issue. And there should be some possibilities to go even further with the idea.

For the record, Zopfli gives (with 1000 iterations) the following:

3986 prod1.runner

And yes, just relaxing the character set to more printable ascii chars, get these:

4076 prod1.runner 6588 prod1.obfuscated.bin 3946 prod1.runner (with Zopfli.)

Now it looks like 170 bytes in 10 hours. Not too bad anymore, and now it has become so much easier to do these changes in the obfuscator. This was a good day (and night) for this hobby project. Unfortunatly for the hobby, there will be a full working day and perhaps many already this week. Will there be enough time before the postponed Assembly Summer 2021? I hope so.

2021-08-12T03:16 (6 hours 11 total)

Some quick playing around with encodings. With 92 char alphabet, either one char or counted string:

4054 prod1.runner (zopfli 3928)

With 92 char alphabet, always 2 chars per word, no separators:

4188 prod1.runner (zopfli 4074)

Clearly not a good turn. Less is less. So.. try with even less..

2021-08-12T22:02 (1 hour 12 total)

With either one char or using 0x01-0x0f as additional initial alphabets:

3985 prod1.runner (zopfli 3870) 6078 prod1.obfuscated.bin

We have a leader in this competition. It's not pretty, but it's small!! This enables using about 15*90 different words which is more than any 4k production source could ever have anyway. Also, initial bytes 0x10-0x1f are still available for other uses.. I think I might come up with a few. Starting with string literals for shaders or something, for starters..

Less is less.

Hmm.. and wow, the original obfuscation is looking quite cool and tight with hexdump canonical format, in all its toxicity.

And now my crunching speed looks like 260 bytes / 13 hours. Not so bad anymore. What to do next to get more space for creating..

2021-08-12T23:07 (1 hour 13 total)

Register-relative addressing of internal variables:

3976 prod1.runner 5979 prod1.obfuscated.bin

100 bytes from payload, only 10 from packed. Absolute addresses did do a good job. But less is less, and the hexdumps become more compact once again. I'll have it like this, and move on.

Make boot-strapped execution tokens register-relative and in kin with internal variables:

3975 prod1.runner 5966 prod1.obfuscated.bin

Reduce 23 bytes from payload and a lousy 1 byte from exe!? Testimonial to the efficiency of repetition in packing. Depending on what we're doing and how much time we have for hacking, it seems to pay off really well to just use self-similar patterns like absolute instead of instruction-relative addresses in machine code. But now I have all the time in the world to shave bytes, one at a time, so this is after making variables and xts really similar to each other, also in the program logic:

3971 prod1.runner 5966 prod1.obfuscated.bin

These were to be tried. Now they have been. Moving on.

And restart the measureable after adding redundant spaces for visual inspection of hex dumps.. It's nice to see clear stripes between the engine, the machine language strap side, and the forth side. Just remember that a final version doesn't need these.. could weigh as much as 10 bytes that you'll want to use for something that actually executes.

Efficiency is dropping again, as I'm getting tired. Gotta sleep and work.

2021-08-13T02:30 (3 hours 16 total)

Couldn't go to sleep. Must. Look. At. Code. And. Minimize.

I am now quite happy with the assembler side. If there are ways to make it more optimal then they are beyond my capacity. It is good now. It is art. It has soul.

I should revisit the note of 2020-12-28 now and make a decision. I think I am ready now, after experiencing the narrow creative space and the need for bytes. It took hours to spot that next place of possible payoffs. I hope it will be the 5 minutes to handle it that the old notes promise. And I hope it will pay as much as it did with the beep triangle.

And now I need to sleep before the work.

2021-08-13T03:56 (1.5 hours 17.5 total)

I'll save my old code comment here as a sentimental memento:

See the "travelogue" notes on 2020-12-28. Sampled a couple of approaches to this part, and the best thing would be to add the following few lines [mov eax, 0x20ffad48; stosd] here and remove some 120+ NEXTs from below. Other things I came up increased rather than decreased the packed executable size on the sample date. That was still with the beep triangle, but now I can make a final benchmark and decision using a more interesting mock-up. (Comment updated as of 2021-08-13.)

And there: Today I finally accepted to take this unholy route of writing magic opcodes on-the-fly. It may not look like much, but somehow it feels like a major departure from the earlier goal of making a principled implementation.

Current situation:

3904 prod1.runner 5752 prod1.obfuscated.bin

And yes, it was a 5 minute operation, as anticipated. Not even counting that time into the project total. Nah, but again I spent too much on writing and not doing. Stop, stop, stop..

2021-08-13T18:36 (0.5 hours 18 total)

Obfuscator pre-applies CELLS FLOATS SFLOATS to numeric literals now:

3894 prod1.runner

Optimized obfuscated string representation and thus rid of defining S":

3859 prod1.runner (zopfli 3777) 5700 prod1.obfuscated.bin

2021-08-14T02:37 (2 hours 20 total)

Strategic pause and cleaning up

Now, it is time for a time-out and a strategic pause: I need to get back to actually creating soon! Otherwise I'll run out of time for that. The past 20 hours of size optimizations gave me 320 bytes of more space to use. It is not much, but it is more than I had before. It is easy to get hooked in micro-optimizations, and seeing how it melts down byte by byte. But it takes time. I'll need to get creating, and I think that will require some technical groundwork not related to size optimizations but rather convenience functions. Things like sync tracking and automatic shader minification..

So, this is important:

- Remember to stop micro-optimizing and move on to creative work for some time!
- At some point, though, try the automatic inlining of single-shot factors. That sounds like a very fruitful optimization strategy.

Perhaps I'll use just one or two hours to straighten out the codes and comments as of the situation that may have improved from the last day of cleaning up. Starting point:

nFIXMEs: 88 nTODOs: 56

Yep, stuff had been done already, and the FIXMEs and TODOs could be just removed. Clearly, there has been improvement:

nFIXMEs: 81 nTODOs: 50

2021-08-14T16:24 (4 hours 24 total)

Enter the loop of entertaining size optimizations

I made a tentative inlining mechanism. Just a few inlines make the mock-up production considerably smaller:

3831 prod1.runner (zopfli 3740)

Wow - I really need to make the inlining automatic to some extent.. There is a lot more to be compressed using inlines than I want to mark manually! Meanwhile, the lack of inlining constituted many of the issues in the obfuscator. So, the source just got just a bit cleaner again:

nFIXMEs: 80 nTODOs: 44

Also: I re-learned how to make classes in Python. When do I learn to do such things earlier? I think the obfuscator would have been easier to develop if I had thought of classes and objects earlier. Idunno.. Maybe I didn't yet have enough complexity to go beyond strings and dictionaries back then. I won't worry about the past. The latest 2 hours very well and efficiently spent! Soon enough I'll be creating again, and I can do that in a much larger space than before. With automatic inlines, there would be even more… and that's not micro-optimization, that is macro-optimization… almost literally.

2021-08-20T00:23 (2 hours 26 total)

Then, total procrastination. Must go to sleep.. 20 bytes saved by micro-optimizing the bootstrap, but at the cost of too little sleep and a very inefficient project hour to be recorded here. This could have been done at any later time, in a shorter time. Why can't I learn away from this shit?

3811 prod1.runner

2021-08-20T01:47 (1 hour 27 total)

Too tired to do anything. So why, oh, why did I even try? I hope there was some subconscious thinking going on, because there was definitely nothing sensible coming out from the conscious mind during these late night hours. Some brainfarts that went straight to the bin.

2021-08-21T02:11 (3 hours 30 total)

Not as tired tonight. I'll count 2 hours of hobby programming up to now, because I think that's about as much as it would have taken without distractions. Did this on and off during commercial breaks of TV shows. Couldn't solve big problems like this, but I did refactor the obfuscator so that we'll be seeing a major reduction in packed sizes in just a couple of minutes from now. As of right now, it is not yet impressive:

3804 prod1.runner

2021-08-22T01:53 (2 hours 32 total)

Yep. I'm getting tired again, so it is difficult and slow. But I was able to make a small benchmark of what should be possible with automatic inlining of stuff (and/or manual, but should be easy to do..):

3617 prod1.runner (zopfli 3527) 5615 prod1.obfuscated.bin

2021-08-22T04:05 (2 hours 34 total)

Some easy bytes shaved from the Forth side of bootstrap:

3590 prod1.runner (zopfli 3501) 5551 prod1.obfuscated.bin

Few minutes of time, hardly "billable". Making the note on a coffee break at work.

Soon I must command myself to stop minimizing and start creating. But, come to think of it, I will need some way of using an automatic shader minifier.. editing shaders embedded in Forth source as S" strings is a big bottleneck in the creative flow. And I'll have to put a couple of thoughts to sync tracking, perhaps with Rocket?

2021-08-24T13:27 (0 hours 34 total)

Further easy bytes, now from the Assembler sisde:

3575 prod1.runner (zopfli 3485)

2021-08-24T14:49 (1 hour 35 total)

I've got space - I need to stop the small tweaks now.

3534 prod1.runner

2021-08-24T22:36 (3 hours 38 total)

Inline mechanism is now sufficient for the current episode of this project. Next up: Shaders using file input, automatic shader minifier run on the obfuscated path.

2021-08-24T23:54 (1 hour 39 total)

Done. Hours can be really efficient when there is plan, focus, and enough sleep. I did borrow some hours from the new students, but they'll get their service, too, before the sun rises tomorrow. For this project, I'm gonna call this one a day, after merging to master and pushing to the clouds. Maybe some clean-ups and benchmarking with the old POC that had more shaders to minify..

2021-08-25T16:42 (3 hours 42 total)

Strategic pause again

No. I started with the older POC codes, and quickly decided it will be wasted effort. Rather, I need to do the following:

- Clean up current mock-up "prod1" source, and handle as many FIXMEs and TODOs as possible. Then leave it and move on.
- Sort out directory structure and automatic release packaging of a compo entry.
- Quickly scout possibilities of sync tracking.
- Just start creating

2021-08-26T22:13 (1 hour 43 total)

While cleaning up comments, I'll copy one here because it is something to be remembered.

For final compressed executables, be it noted that the first two factors below (i.e., SF>ARR F^-1 actually increase the packed size by many bytes. I find them extremely useful for reducing noise in source code, but I would suggest (to my future self) to have an :INLINE version of : or an "[INLINE] :" idiom that would make the obfuscator replace word uses directly with the contents of the word.. Like a true "compiler macro". Might end up with balance between code readability and packed size. The third factor (array swap) instead reduces the packed size by 20 bytes for only 3 applications of the word. So, it depends. And therefore, it would be beneficial to make it easy to play around with inlining / not inlining of words. The inlining would be useful for one-time-use factors such as InitApp FinalizeApp RenderLoop and the sort.. but I don't want to break the Forth statefulness by trying to automatically guess if inlining should be done or not. Let it compile on-the-fly, like Forth is meant to do…

With further clean-ups and re-organizations of code, the metrics have come just a few steps down again:

nFIXMEs: 76 nTODOs: 45

2021-08-29T01:52 (3 hours 46 total)

Just had to try one of the benchmark ideas of some earlier day.. Here's the comment for the records:

So, I did the benchmark, and the packed exe actually got larger when some of the float convenience routines were taken back to the assembler side. So, let it be, and go on with other things. One longer word was better in assembly, and I got a pathetic 10 bytes of space. Oh, joy. This hour didn't give improvement, but it gave knowledge.

3524 prod1.runner

2021-08-29T03:07 (1 hour 47 total)

Enter the loop…

Some futile hours of late night attempts.. And then a couple of good afternoon hours. Made some quite promising factors in the synth vocabulary, as a step towards easier sound generation. The factors come at the cost of 10 bytes, which is peanuts compared to the clarity they bring to synth representation. I put back the snare drum. It costs some 40 bytes but that includes the random generator which is probably going to be used anyway.

I think the sound lexicon started going to the right direction. Soon I'll have to start cooking up a song that could potentially be used in the final production, even. Time is running out very soon. Basically just 80 hours left!

2021-08-31T21:30 (3 hours 50 total)

Evolving it. Just moving forward with the audio lexicon. Good pace, nothing special. By the way, the prod1 mock-up is still good for benchmarking size of the lowest level bootstrap. At the moment:

3564 prod1.runner

It got some bytes fatter with the randoms.

2021-09-03T00:16 (2 hours 52 total)

Well, well.. Assembly Summer 2021 got cancelled. No more pressure from that deadline. On the other hand, it is a bit of a hit to the motivation. This thing has been too long in the making already. I was prepared to put this out in whatever shape this was, in a month or so, but now I guess I'll have to keep on just honing and honing.

So.. slow down and experiment with different alternatives.. there is no hurry now. Which is good, in a way.. I'm 2 ECTS into the final episode already! Only 3 more to go before wrapping up at least this learning journal. The production itself… will be worked on until the final minutes before deadline, likely in August 2022, no doubt about it.

What next then? I'll prepare a couple of shaders, and then think once more about sync tracking and sound.

2021-09-07T02:00 (3 hours 55 total)

Some obvious micro-optimizations to the assembler side.

3559 prod1.runner

Three hours not good in metrics. But I'm getting TODO-items closed.

2021-09-12T03:11 (3 hours 58 total)

Some nightly developments:

3550 prod1.runner 3541 prod1.runner

Back in business.. after a month, it seems. That's life, I guess.

Estimating some 3 hours of project time from the undocumented "night" and this very nice Friday evening.

2021-10-08T22:25 (3 hours 61 total)

So, what was I doing a month ago? Luckily I have comments. I'll try to pick it up. But it's gonna be easiest to land by reviewing and fixing a few outstanding issues.

It would be really neat to have the assembler side polished without any TODOs of FIXMEs.

2021-10-08T23:32 (1 hour 62 total)

And there it went away - the last FIXME of the Assembler side. Six TODO-notes remain. Let me rid of them, as a warm-up after month of not touching anything.

3522 prod1.runner

Some 19 bytes shaved by sorting out one TODO-note.. Thank me in the past for making such notes…

2021-10-09T01:27 (2 hours 64 total)

Getting things done - slowly, slowly, but done.

2021-10-09T04:49 (3 hours 67 total)

Just to feel happy after a night of coding.. some measures are now smaller than before:

nFIXMEs: 74 nTODOs: 38

Some of the old comments start to look funny, even. An example: "FIXME: Is this proper usage of DOES> ? Is it really as simple as this?" (in the DOES> part of FCONSTANT definition, to have it F@ its contents when executed)

So funny.. much more so than back in the day I wrote it.

Ya-ha.. and then.. I'll kill some more darlings as soon as possible (from the file appropriately called darlings1.fth)

But remember: Clock is still ticking, in a way.. I need to get something "cool" finished in just a few dozen project hours from now. Try to avoid the procrastination limbo..

2021-10-09T06:14 (1 hour 68 total)

Didn't kill the darlings yet, but demoted the urgency of their backlog notes.

nFIXMEs: 51 nTODOs: 33

But again very inefficient. It's the same old procrastination monkey again.

2021-10-10T03:10 (1 hour 69 total)

Outcome of tonight:

3518 prod1.runner nFIXMEs: 49 nTODOs: 31

Doesn't look like much progress at a fist glance, but, observe: The assembler side is now completely cleared of TODOs and FIXMEs. It can be improved, as everything can always be improved, but all the issues raised during the hundreds of hours of project time have been dealt with. It's clean, mean, beautiful, and dangerously alive..

2021-10-12T03:51 (5 hours 74 total)

3513 prod1.runner

2021-10-12T22:09 (0.5 hours 74.5 total)

Metrics down, down, down.

3512 prod1.runner nFIXMEs: 45 nTODOs: 31

2021-10-13T01:20 (0.5 hours 75 total)

Finally back to the creative side. Better late than never. Full speed ahead. Plan:

- Try some delay effects and LFOs to make sound more interesting (with little data)
- Revisit the cube asset and shaders to enable true wireframe cubes.
- Try some usual post-processing, like bloom
- Try some “older tricks from the books” like visualizing the sound wave being played..

There is not much space for being creative, but I can’t keep on just minimizing - I need to see and hear something more interesting than the current mock-up which is really dull.

2021-10-14T00:45 (2 hours 77 total)

Filling up the space again: synth lexicon and learnings on DSP

Hmm… learning stuff from https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverberators.html

Wow - it was surprisingly easy to implement a “Schroeder Reverberator” that sounds quite cool out-of-the-box (a box which is 50 years old… a very suitable age for this Forth project; I hope this algorithm could be a keeper in the production.. even if the first version fattens the exe by 200 bytes which is just too much.)

The road from having just a crude delay to having a thick reverberation: 1 hour. Things can develop absurdly fast as well as absurdly slow. This night was good. Must sleep and go to work, even if I’d like to just toy around with the reverb now.

2021-10-15T01:18 (1 hour 78 total)

Classic.. Missing sleep because I just had to stay up and play with it. Oh, the joy of new toys, self made.. With the delay and reverb, the audio can be easily made non-boring and retro-techno-allusive with very few notes actually played. But I’m facing a major refactoring of the synth side before I’m there.

But now I have a musical atmosphere that already alludes to the right directions. I think it will bring about an inspirational mood to the trying out of the graphics things.

Now - a new try to get hands off this one.

2021-10-15T02:48 (1.5 hours 79.5 total)

On and off on a Saturday afternoon, 3-4 hours in total maybe.. I'll say approximately 3.5. Implementing stuff that has been preparing in the back of the head every now and then.. so they mix, project time and calendar time.

2021-10-23T23:57 (3.5 hours 84 total)

Almost unbelievable.. I think I make it shorter and cleaner all the time. Yet, the packed executable seems to be getting bigger and bigger. A few bytes at a time, but still it's getting bigger and not smaller as I'd expect. Bloaty as it might be, the synth and sequencer code is now becoming quite easy to customize.. Not yet completely there, but soon.. The synth is now a completely new story, nothing like the one in the early mock-up "prod1", so I'll need to start benchmarking the newest production. It is now 4.1 kilobytes:

```
4122 test_by_me/test.runner
```

The synth weighs a lot at the moment.

2021-10-24T05:02 (3 hours 87 total)

Yep.. thinking smaller but becoming bigger. Funny. But it sounds better again. More aesthetic. Cleaner code, too. I'll optimize the size later. First, I'll get it to sound right, and look cool in the source code.

```
4136 test_by_me/test.runner
```

2021-10-30T02:29 (3 hours 90 total)

The synth is becoming much more capable than even necessary to get some gloomy allusions and a not-boring-to-death soundtrack. It will need downgrading at some point. I think I'll put in just a few more features.. things must be built before they can be torn down, right?

```
4164 test_by_me/test.runner
```

2021-10-30T15:09 (1 hour 91 total)

Some nightly night coding. Never very efficient, but these circumstances are what they are. Just no time for daylight coding with a clear head.

2021-10-31T01:59 (0.5 hours 91.5 total)

On a train, returning home from Assembly Creative Tech '21. Great party, great people. I decided to not even try to make a compo entry. There would have been no time anyway. But wow, what a nice event it was. I'm really looking forward to face-to-face demo parties once again.

Micro-optimized some things, pretty much for fun - think sudoku or crossword puzzles. Shaved some 10 bytes:

```
4152 test_by_me/test.runner
```

2021-11-06T18:42 (0.5 hours 92 total)

Some more byte melting in the train:

4139 test_by_me/test.runner

2021-11-06T19:35 (1 hour 93 total)

Commit yesterday's train coding.

2021-11-07T22:37 (0 hours 93 total)

Some ideas, better and worse, tried out.

2021-11-08T23:38 (0.5 hours 93.5 total)

Yet again, the usual 6 weeks completely without a thought. Use this situation as a booster: Now that the brain has been re-set and it has forgotten what it possibly was stuck with, I should be able to prioritize the next steps. Here goes:

- Implement INCLUDE to make everything nicer.
- Bitmap fonts for debug purposes and for fun. Perhaps for production, if the size can be reduced enough?
- Decide on graphics primitives and main visual elements

2021-12-29T14:51 (0.5 hours 94 total)

Ok… I tried out the INCLUDE, and remembered why I didn't take it up earlier. It is too much effort that is orthogonal to just doing a creative production now. Polish the system later, if that later ever comes. For the time being, decide: No INCLUDE is available for this production. I'll keep catenating source files externally.. the build pipeline is a mess, but it is already there.

I wonder if it will be as difficult with the fonts.. Let the brain remember all the difficulties once again, after the break…

2021-12-30T00:18 (3 hours 97 total)

Trips to fonts and inspiring lofi games

Added the possibility to load a bitmap font. The first try with a complete 256-character font collection bloats quite a lot:

6794 test_by_me/test.runner

Hehe. I'm using a downloaded set of Atari ST bitmap fonts - a blast from my past. I'm not adding any fonts to the code repository yet to remain paranoid about copyright laws. Bitmap fonts should be beneath the law. But I'll stay safe. Any 8x8 bitmap font can be downloaded and used as the 'sysfont.fth', depending on your own nostalgia… Atari, Amiga, and IBM PC system fonts seem to be easily available in the Internet.

It should be easy enough to paint some 8x8 pixel fonts of my own if necessary. First, I'll need to see how many characters can be used, if any… If it turns out that fonts are used just for debugging, let us roll with the nostalgic choices available as external downloads.

2021-12-30T02:43 (2 hours 99 total)

Not inlining ‘0f,’ and ‘1f,’ saves 460 bytes from the packing right away:

6334 test_by_me/test.runner

How about the funny alternative of using the same (re-defined) words for pixels and synth note onsets? I’ll try that just for fun, later.

The obvious and necessary thing is to reduce the number of characters, of course..

Back-of-the-envelope fast tries. Remove unprintables 0-31:

6070 test_by_me/test.runner

Then remove those beyond ASCII..

5087 test_by_me/test.runner

Zopfli with 40 iterations can get all ASCII printables to a pretty nice package:

4756 test_by_me/test.runner

Ok. . . so re-defining O and _ which are also onset and pause in music gives:

6330 test_by_me/test.runner

Also, now ‘0f,’ and ‘1f,’ can be inlined in their normal places:

6326 test_by_me/test.runner (zopfli 5647)

And with only ASCII:

5038 test_by_me/test.runner (zopfli 4716)

With almost only A-Z and a-z:

4766

With exacly . ’ A-Z and o and k:

4554 (zopfli 4426)

Absolute minimum character set that can write “. . . . . . ok ’ DEMO EXECUTE ok”: 4425 (zopfli 4316)

Maybe workable? Locations in bitmap were not yet optimized. Hmm. . . the bitmap data of .ok’CDEMOTUX fits in 200 bytes in fact. Base with all but data is 4233 bytes.

And it will get a bit smaller if there are no empty encodings. How about just a bitmap image that contains the "ok ’ DEMO EXECUTE ok’ phrase?

2021-12-30T04:37 (2 hours 101 total)

I just played PICO-8 games for 5 hours, and I’m counting these hours as efficient time for this project. I just had to see some stuff in 128 by 128 pixels because I need to decide on my lo-fi screen resolution. Right? I also enjoyed the games quite a lot… PICO-8 is cool. I might want to try making something on it at some point. Oh, well, I need to get this toy of my own finished first.

2021-12-31T04:57 (5 hours 106 total)

Looked into ALLOCATE RESIZE and FREE of the Optional Memory-Allocation word set. Could be useful to be able to ALLOCATE stuff dynamically instead of always ALLOTting. Maybe.

Sorted out the current mock-up graphics code. Next up: creative toying with the graphics while making the graphics lexicon leaner and cleaner if possible.

2021-12-31T14:38 (1 hour 107 total)

Happy new year. Too many years into this project already. I hope this will be the final year. Get this out of the system and start with something else.. possibly re-do from start with Lisp… One thing at a time. Now it is Forth time. Not so many days off work, so must try to get some hours pushed into this project.

Just pushed two hours of project time. Pretty much just having fun on a holiday night after things got quiet around. Designed my own system font which is 5 pixels wide, 6 pixels high, and made by myself from scratch. Copyrighted or not, I have a system font of my own now. Committing to git right away as afont5x6.fth …

2022-01-02T03:34 (2 hours 109 total)

Looking further into fonts. Created afont4x5.fth with two alternative storage formats. The work, the work… it is beginning after holidays. But I try to use a couple of hours each day for this project now… I basically have only a few dozen hours left anyway.

2022-01-04T13:26 (2 hours 111 total)

And then back to the loop of micro-optimizations

A very good day of hobby coding. These holidays are too short. Just a quick note on developments: Revisited the OpenGL lexicon and implemented ALLOCATE to enable creations during run-time of asset initialization words. The application code looks a lot nicer and cleaner now, in my opinion, so I guess this was necessary. The size benchmark looks quite disappointing again. The momentary reference version went from 4633 up to 4690. This is the same old story once again: I thought that the increased simplicity would make it both easier to maintain and smaller in size. But all the time it seems that the smallest exe comes from just hardcoding and inlining everything.

So, this might be a round-trip: Use these bloated but easy-looking things while developing, and then eventually come back to hardcoding the hard way, to make it small.

2022-01-05T21:15 (7 hours 118 total)

And a very nice night of hobby coding following the day. Too soon back to work. Too soon at the deadline of this project. I'm gonna do a little more hours than allotted. Pay the price in the evaluation then. But these projects should be for fun, first and foremost, and I just did have many hours of fun. Priorities! Now sleep and proper non-computerized holiday things for the day-off at Epiphany ("loppiainen" in Finnish).

2022-01-06T02:56 (5 hours 123 total)

Five good coding hours in bright daylight, thanks to holidays. The application code is now much cleaner and nicer than before. It is no surprise that the executable size increased by 100 bytes again. I'm at 4790 bytes with what was 4633 before making it look nice. Anticipate a round-trip back to something quite awful when finalizing an actual 4k entry.

That, by the way, should happen quite soon. This episode is soon at the magical 135 hours checkpoint.

2022-01-08T00:31 (5 hours 128 total)

The disappointments, blah. 4744 after things I thought should make it smaller again. Damn. All kinds of flexibility and fun is just extra weight in bytes.

2022-01-08T04:31 (2 hours 130 total)

This and that; still size-optimizing. Coding like it was the last night of the holidays.. which it pretty much is, unfortunately. Overdue. Must get to the creative part, understanding that I probably can't fit in every element I originally wanted. Especially fonts weigh a lot. And also the current synth. And the neat wireframe stuff.. Gotta let go of something. Try to pick the one or two most essential elements, cry a bit for the loss of everything else, and get creative with the few things that remain!!

2022-01-09T04:12 (2 hours 132 total)

Back to this, after starting the spring at the uni. Wow - I really seem to be out of time for this episode. I shall be self-evaluated like the couple of my students already working under the same self-assessment template. We all need to aim at the 5 ECTS / 135 hours scheme.

Time is up. Decide: Use maximum overtime for this "course", and take the resulting hit in grade assessment. Now, knowing that the time is up… really focus on JUST, ONLY getting a creative product down.

Ergo:

- Ditch fonts and text and maybe sound effects. Kill the darlings to fit 4k.
- Force creativity: Start from scratch, alternate work between the "post everything" shader and the lofi content quickly to produce possible tentative concepts from scratch. Stick to the original aesthetic plans (minimal, triangles, Jarre, etc..).

- Main thing: Kill the darling, start from scratch. Even if time is practically up already.
- Re-iterate: Start from scratch.

2022-01-30T03:05 (2 hours 134 total)

Bang - it's all overtime now.

According to plan, I tried leaving out the reverb effect and the sequenced song.. Turns out it is not much of a win.. The whole audio thing currently weighs only some 500 bytes. That much should be possible to save from other places.

So, back to the drawing board. Stick to the plan:

- Kill darlings and start from scratch, but focus on things other than audio.
- Consider the audio system frozen and double-checked to be a keeper for this production.

2022-01-30T20:06 (2 hours 136 total)

A great coming to senses, with a feeling of acceptance

Almost 3 months of not touching this, it seems. There's only so much time for hobbies, and I've been giving a deliberate bias to jogging. I think jogging might have a better long term impact than hobby programming. Even in short term, it allows the working days to be more efficient.

Today, right now, I'm watching the live stream of the Revision 2022 demo party, unfortunately unable to be at our local satellite.

They are showing the PC 4k intro competition at the moment, and the productions are… well, astonishing, again. Is this really the competition in which I want to submit my humble Forth system and supplementary audiovisual demo? Against these competitors? Yeah… I'll have to do a lot of more work, so it is best I don't try to rush things. My eyes are in the Assemby Summer 2022 now, so I have a couple of more months. Three more months of not doing anything won't be an option, though.

After a thorough re-thought, while jogging, I've decided to do the opposite of what I accidentally thought in January: I'll wrap this episode up without any more extra hours. It will be a magnificient and exemplary failure.

The Revision 2022 stream has gone to 256 byte intros in DOS-box. Oh, wow. I want to do that one, too, in the near future! But, one thing at a time, the next thing I really want to do is to take part in the 4k category of Assembly Summer with this current thingy.

A worrying fact from Revision 2022 compo rules: They say 'It is not allowed to use code-shortening techniques that exploit "pattern matching" heuristics over the items in an API to avoid referencing their names in full. For example, iterating over WebGLRenderingContext or Canvas2DContext objects to insert function aliases based on a few selected characters of the name is not allowed. These tricks are highly likely to break in future browser releases as these APIs are extended over time. Also see https://www.pouet.net/topic.php?which=12276'

Oh, bummer, this is exactly what I'm doing with the SDL2 and OpenGL APIs here with the Linux 4k intro target. It is not on "browsers" here, but basically it is the same name-hashing trick on Linux. I learned it from people who used it before. Is it going to be banned from now on? I think this should be debated on Linux platforms, if a competition should allow participation on Linux in the first place… Things are pretty much fixed on API versions (like "something.so.123"). In my opinion, "Works on the compo machine" should be enough to ascertain that the size limit has been met on the match night. And then share the source code, if it is necessary to be able to run the production after the competition.

I hope they will have sensible and fair rules in Assembly Summer 2022, with a possibility to participate on Linux.

Oldskool Demo of Revision 2022 already going on in the stream.. pretty interesting stuff they have made on so many old platforms. What was a WonderSwan anyway? Ok… live and learn, now I know.

Beautiful stuff in the Revision 2022 low-resource compo categories, very inspiring.

As for me here - I'll conclude this learning episode with exemplary notes of an exemplary failure. As soon as I get back to this. Right now, I'll just watch the Revision oldskool entries and not count it as time spent in this project.

2022-04-17T01:47 (2.5 hours 138.5 total)

(just experiencing the remote party via stream…)

2022-04-17T02:15 (0 hours 138.5 total)

(another night of watching the Revision stream…)

2022-04-18T02:00 (0 hours 138.5 total)

Ok, let me just copy-paste my latest and greatest self-assessment template here once more, and fill it in.

It seems that I had to have a total blackout for this project once again during the busier part of the academic year. A fact of life, it seems.

Well, now it is more or less holidays, and the Assembly 2022 happening on-site. Maybe I get this out now.

I spent approximately 1.5 hours getting back to this once again, reading these notes of the 11 past months of calendar time. Not much time could be spent here. No more do I understand how a student can get a degree while working full-time. Well, not many of them do, and now I think I have here a very good first-hand experience of what it could be like for the majority that don't.

2022-07-01T02:00 (1.5 hours 140 total)

Blackout

De-ja-vu: Read through the earlier notes. Seven months have passed. I have done a lot of things, none of which was this project. Three students have started work under my self-assessment, two have finished.

2023-01-30T01:29 (1.5 hours 141.5 total)

The most spectacular and commendable failure

The template comes from my notes5.txt - this is the second trial run using it.

1. Time used for this 5 ECTS learning episode.

   Must be tracked and documented from the very beginning to the very end, at least on the detail level of "what I worked on, when, for how long at a time". Tracking for example "thinking time away from keyboard" is very fine, when you have thought about the issues of this course. Use common sense. Find a convenient granularity for yourself. I have tried to provide one example in my demoscene hobby projects. But make it in your own way, as long as the above minimum requirement is met, and a total hour count can be observed.

   0==FAIL: less than 70 hours (reeks of no interest to the topic or the whole field of study! Stop this madness, do yourself a favor, and find something more interesting to do with your precious one life than this kind of topics!) OR no documentation provided (won't fly; redo from start, with time tracking!)

   1/5: 70-81 hours (about 50% of what was actually expected. Absolute minimum! Rethink motivations!) OR more than 168 hours (one week or more of extra work. STOP. Start next course instead!) 2/5: 82-94 hours OR 160-167 (four full working days, almost a week. Hands off already!) 3/5: 95-108 hours OR 152-159 (three full working days too much, almost time for a 1 cr course) 4/5: 109-121 hours OR 144-151 (two full working days extra time is a bit much "finalization") 5/5: 122-143 hours (target is 135; one full working day extra finalization is appreciable.)

   MY GRADE: 5/5 if I get this evaluation done before 3 am.

2. Ability to find focused time, "getting into and staying in the Flow"

   0==FAIL: (impossible to fail because of this category)

   1/5: I basically failed in allocating focused studying time - I had lots of distractions all the time, and I had very short periods of staying on topic and actually working towards the course goals. 2/5: Between 1 and 3 3/5: It feels as if 50% of the hours were well spent and focused on the task at hand 4/5: Between 3 and 5 5/5: Almost all, say 95% by gut feeling, of the hours were spent without distractions, moving towards the goal.

MY GRADE: 1/5 because of very long times away from the whole project, and a lot of wasted time in trying to re-start, but not really making the necessary time available in the daily calendar.

3. Technical requirement achievement

   Must be verified by the actual code in question that is shown to the teacher along with this self-assessment.

   0==FAIL: I was unable to produce working code for 50% or more of the required tasks. 1/5: 50% of required tasks produce expected results for clean input with my own code 2/5: 70% of required tasks produce expected results for clean input with my own code 3/5: 80% of required tasks produce expected results for clean input with my own code 4/5: 90% of required tasks produce expected results for clean input with my own code 5/5: 100% of required tasks produce expected results for clean input with my own code

MY GRADE: 2/5 maybe? How to interpret this now that there was no predetermined set of exercises? I was aiming to a compo-worthy 4k executable with cool audio and cool graphics. I'd say I did build most of the audio side and some building blocks that could be used in final graphics. It is still far from a complete production. Let's say, optimistically, that I got to 70%. The core of the obfuscated Forth system got much smaller which should be counted as favor even if this was already supposed to be the "final practical". But maybe I'll account for this in the following sub-grade, "technical quality".

4. Technical quality achievement

   0==FAIL: (impossible to fail this category)

   1/5: Poor quality: My code is not commented, and it would be difficult for other people to understand. 2/5: Between 1 and 3 3/5: Good quality: Another developer could pick up my code and both use it for their own applications or develop it further. My comments explain all of the API and the reasoning behind crucial choices that differ from those expected by an application programmer. 4/5: Between 3 and 5 5/5: Brilliant: I am proud of my code. I feel confident that I could continue its development easily and that it could be used easily by an application programmer by reading the documentation / examples that I have included.

MY GRADE: 4/5 I'm proud of many things. Yet, it doesn't meet "easy adoption" by others, I suppose.

5. Learning achievement

   Must be explained in words: How and why did you learn or not learn each of the topics that were part of the course curriculum? (The specific list of curricular topics could be copied here for convenience on an actual course)

0==FAIL: (impossible to fail this category)

1/5: Several concepts remain vague: I find it hard or impossible to give examples of what I learned of the intended subject matter. 2/5: Between 1 and 3: I was able to differentiate between topics learned and not learned. I gained skills in some of them. 3/5: I learned a lot: I know what each topic of this course means in the big picture. I know some of the topics so well that I could use them at work or teach them to others. 4/5: Between 3 and 5 5/5: Brilliant: I left no stone unturned. I master every detail on the level required in the target skill description.

MY GRADE: 4/5, mere modesty to not say 'brilliant' here.. reverberation effect, legal status of bitmap fonts, more and more Forth, … so much learned again.

6. Learning extent and zone of development

   Grades 2-5 must be explained in words: Which were the specific things new to you.

   0==FAIL: (Impossible to fail this category))

   1/5: I learned nothing new; I only applied things I already could do 2/5: Between 1 and 3 3/5: I learned a lot of new things (which can be mandatory elements, if those were new to you, or voluntary challenges like a new programming language, library, syntax, design pattern, development tool, etc.) but mostly I stayed in my comfort zone with familiar tools and methods. 4/5: Between 3 and 5 5/5: I made sure that I don't do things "the old way" but try to be working with something new to me all the time (95% by gut feeling).

   MY GRADE: 3/5, a bit difficult to understand the nuances between "learning achievement" and "learning extent". In this project stage, I must say I didn't really go beyond the usual and the familiar. New, yes, but in a pretty safe zone.

7. Look back and forth

   Obligatory part, but not graded

   0==FAIL: Missing

   PASS: Each of the following questions are answered.

   7.1 Reflect back: What was the easiest thing for you to learn or do?

   MY ANSWER: Schroeder reverberation was surprisingly easy to understand and implement.

   7.2 Reflect back: What was the hardest thing for you to learn or do?

MY ANSWER: Look, I totally failed to spare time in this hobby project, which was supposed to happen. This is the one thing to be learned. Didn't learn yet.

7.3 Reflect back: What was the most useful thing that you learned?

MY ANSWER: Scheduling time for hobby projects is difficult.

7.4 Reflect forward: Where do you plan to use the things you learned here?

MY ANSWER: I did learn this and that about basic sound and graphics. Will be used in my classes, as practical examples of relevant theoretical subjects.

7.5 Reflect forward: What would you like to learn next about these things?

MY ANSWER: I think the learning is now done, and next it will be a matter of "just doing it", the production show.

7.6 Reflect forward: How and when do you plan to do that learning?

MY ANSWER: Looking back at how well I have been able to live up to the plans and the promises I hace made to myself so far, it is better to stop planning. I will just do it, or not, … it might mostly need a decision now rather than a plan.

7.7 Reflect forward: What will you try to do better in your future studies for things to go smoother?

MY ANSWER: Really, really tough question. I think the main improvement is to not plan or promise - just do (or not).

8. Overall grade and plea for adjustment

   Basis of your grade is the arithmetic mean of the six graded sub-areas above. Round to the nearest integral number.

   If any of the sub-grades are 0, or any required verbal explanations of the self-assessment template are missing, you will have to do more work to complete the course.

   Compute the mean here: (g1 + g2 + g3 + g4 + g5 + g6) / 6 = (5 + 1 + 2 + 4 + 4 +3) / 6 ~ 3 (rounded)

   Finally, if you feel that the mean is not a justified grade for you (in either direction), you may plea for adjustment of +1 or -1, with the reasons you feel justified.

   Optional plea: (MY ANSWER) There is an urge to plea for -1 because I have been a "bad boy" and not really working with this project, most of the time in 2022. And I could have done a much better job in focusing the efforts of late 2021 to just finish this. I guess I would make that plea as a student. As a teacher, I would not accept it, because there have been obvious improvements and it doesn't really help to beat yourself. Sort

> of… try to feel good about the learnings and try to get a positive flow in whatever you choose to do in the future… Be happy with this episode being wrapped up.
>
> The teacher reserves all rights to either accept or decline your adjustment plea and to decide your final grade in the whole 0-5 spectrum, adhering to principles of equality and other relevant laws in effect in the EU and Finland. If the grade differs from your self-assessment, the teacher will give justifications to the decision.
>
> Remember that dishonesty in self-assessment is a serious offense, and will lead to investigation and the most severe punishments possible if and when detected! It is very important that the claims you made about your exercise achievements can be verified also in the source codes produced as part of your course work. For example, a copy-pasted tutorial code without your own comments hardly warrants you to say you have learned a thing. The situation immediately changes for the better if you have tried to build the code from scratch first, using reference manuals and only the narrative of the tutorials, and then describe a controlled path of discovery from your own attempts to the fully-working examples available in tutorials.

That's it then… This one is wrapped. If it was a real study unit, I guess I would be happy that I got a good grade and supportive feedback. Then… to decide what I want to do with this topic next… obviously I havse no options. I'm gonna keep working and publish, no matter how long it takes. On to the next episode, even if it was never the plan to go that far…

2023-01-30T02:39 (1.5 hours 143 total)