31 January, 2022

Ford C Programming Academy 2022

Progress Summary - Week 03

During week 3 of training, we continued to make progress on a number of fronts, including Git, numeric and structured types, interrupts and MISRA compliance Students completed the following group and individual programming assignments:

"Put it in a Box"
Unions and enums are used together with structures to produce a family of fixed-size types that could successively occupy the same space in memory.

Taylor Series expansion of sin(x)
Loops, counters, numeric types and the limits of numeric algorithms.

Generation of prime numbers in an infinite loop
Used as a basis to recap header files, and to introduce signals and software interrupts.

On Monday, we covered how to create repositories in Git, how to commit changes, and how to push/pull from remote repositories on GitHub. In addition to command-line support with git-bash, we used Visual Studio's integrated support for Git. On Tuesday, we covered the important conceptual memory regions of the stack, heap, text and data segments. On Wednesday, we reviewed and discussed how we could make our code safer and more expressive. On Thursday, we covered automotive safety as defined by ISO 26262/ASIL, MISRA and AUTOSAR, with a emphasis on specific MISRA rules. Friday morning, we looked at interrupts and low-level numeric representations. Later, we examined MCU datasheets and discussed the memory, networking, and hardware-centric interrupts of embedded systems.

Overall Summary

Over the last three weeks, the students and I covered a lot of ground. This was a great group to work with. All participants became more comfortable with the programming environment and its tools, including Visual Studio, Git, GitHub, the command-line shell and compiler. Everyone contributed to the class discussions and stayed attentive to the end. Everyone was able to write programs with loops, boolean logic, arithmetic types, library functions, and limited I/O. I would commend everyone in the class for their effort, attention, and excellent contributions to our numerous discussions.

Right now, all of the students (with the possible exception of those who already pre-tested as proficient) are wrestling with the usual special problems of C: pointers vs. arrays, strings, structured types, and the twisty & and * operators. It takes time to adjust to the language at this level. Everyone needs more practice. Several people expressed interest in an end-to-end example, but I could not illustrate that along the MATLAB/Simulink/code generation/embedded C/deployment axis. Perhaps some in-house training could address that need.

We covered most, but not all, of the objectives initially laid out in the proposed Syllabus. In particular, we were not able to cover small fixed-point representations or bitwise manipulation of values. The proposed coverage of testing was too aggressive to really pan out, and could use another week on its own. A future offering of this class should tighten up and rearrange the materials to allow a better look at binary representations.

As a side note, it occurred to me in the last week that a special challenge exists for engineers learning to code in C, and that is our implicit understanding of numbers. As engineers, we think of them as being real, decimal, floating-point, limitless, and precise to any desired degree, as they are when doing long division or multiplication. Compatible with that understanding, integers are simply a rounded-off subset of real numbers. Languages like FORTRAN and MATLAB do their very best to preserve this abstraction, or even expand it into the higher mathematics of calculus, arrays, and MBD.

Embedded programming and the C language reveal a binary representation of numbers which has disturbingly different properties - it is bounded, limited, base-2, and precise only to a point. Integers overflow and wrap around, encodings are two's-complement or other strange beasts - even the "sign bit" doesn't simply flip a value from negative to positive all by itself. Expanding the Taylor series to more terms is not ever-more precise - in fact, it becomes unstable and unusable using IEEE floats.

Apart from that, numerical analysis emphasizes that mathematical operators are the most important constructs for producing "answers". Using bit- or byte-centric operators, addresses, functions that deal in block memory transfers - even NULL-terminated strings - seems unnatural or contrived. In truth, it IS. Machine ways are not our ways. Sometimes the bits represent numbers; sometimes they are binary switches. The next time we run this course, I think I might try to emphasize that up front.

Paul Kimball
Instructor, FCPA 2022