Fifty Years of Programming
Twenty Years of STL

P.J. Plauger

Dinkumware, Ltd.

My Careers

- Physicist [1961-1969]
- Programmer [1963-present]
- Technical writer [1971-present]
- Science fiction writer [1973-present]

But mostly I’m a programmer.

One Programmer's Progress

- [1961-1965] Princeton undergratuate
- [1965-1969] Michigan State University graduate student
- [1969-1975] Bell Laboratories member of technical staff
- [1975-1978] Yourdon Inc Vice President
- [1978-1988] Whitesmiths, Ltd. President
- [1988-1995] Independent writer
- [1995-present] Dinkumware, Ltd. President

The Physicist as Programmer

- Rule 1: Don’t.
- Rule 2: Get someone else to do it.
- Rule 3: Get the tricky bits from a library.
- Rule 4: If you must do it, make it reusable.

You don’t have to be a professional programmer to need to program professionally.

A Program Product

A program product consists of:

- The code itself, written to be readable and maintainable
- Documentation, for various audiences
- Tests, to demonstrate correctness and to catch maintenance errors
- Scripts, to automate the production of all of the above

Remember, grep is your friend.

Progress in Programming Languages

- Fortran 2, assembler (spaghetti code)
- C (code and data structures, low level access)
- sh (scripting)
- C++ (classes)
- C++98 (exceptions, templates, STL)
- C++11 (template metaprogramming, threads)
- C++14 (more stuff)

Programmers typically program at one to two levels behind the latest language.

Standard Template Library

- Operates on sequences of elements
- Walks sequences with five kinds of iterators: output, input, forward, bidirectional, and random access
- Includes containers that manage sequences with various speed/complexity tradeoffs
- Includes gazillions of algorithms for manipulating sequences

Result: STL is a major advance in reusability.

Template Metaprogramming

- STL originally used limited TM, mostly for dispatching on iterator type (iterator tags).
- TM lets you dispatch on practically any compile time expression (enable_if), at considerable cost in complexity and readability.
- Concepts will let you constrain templates up front, with many fewer unreadable tricks, but they didn't make the cut for C++11/14.

TM is currently hard to learn, but improving rapidly.

Floating Point Arithmetic

FP is supposed to aid the innocent, but:

- Finite precision can lead to significance loss.
- Finite range can lead to overflow/underflow.
- Representable values are unevenly distributed.
- Sensitivity can make some functions not worth computing over certain ranges.

FP code is often not as reusable as it appears.

Special Math

A spinoff of C++11 is an IS called Special Math.

- Includes ellipticals, Bessels, Laguerres, etc. in all three FP precisions
- Proved very hard to write for full range and with (known) good precision, but Dinkumware did it
- Can improve code reuse considerably

See ISO/IEC 29124:2010.

Rules for Numerical Programming

- If you're near a singularity mathematically, you're at it on the computer.
- If you subtract two FP numbers that are nearly equal, you will lose (maybe lots of) precision.
- An iterative algorithm that must converge theoretically might not do so numerically.

Always write a modest number of tests over the ranges of interest, and rerun them after any code changes.

Computational Complexity

- If you know complexity matters, first see if STL has the tradeoff you need.
- Don’t worry about theoretical complexity if the code is plenty fast enough.
- Computers get ever faster, but exponential complexity gets you sooner or later.

Testing

- If you can’t test it, what’s the spec?
- If you don’t test it, how do you know it’s right?
- If you don’t retest it, how do you know it’s still right?

Remember, however, that testing reveals only the presence of a bug; it can’t promise the complete absence of bugs.

Test Suites

- Writing tests is an art that few programmers master well.
- Test for the common cases, the corner cases, and some places in between.
- Don’t test so much that it takes too much time.
- Automate testing so that failures are obvious and easily traced to their cause.

Documentation

Documentation comes in three major flavors:

- Tutorial documentation gives overviews, sample uses.
- Mastery documentation teaches how to use the principal features.
- Reference documentation describes all visible features.

Jones's Second Law – If it's not written down, it doesn't exist.

Rules for Technical Writers

- Use declarative sentences, in the active voice (KISS).
- Avoid jokes, slang, and fancy phrases.
- Omit needless words.
- Say the same thing the same way every time.
- Document it before you forget it.

You’re probably a better writer than you think you are.

Kernighan’s Laws for Programmers

- Write the simplest prototype you can.
- Let your customers tell you what to add.
- Put off the complex bits as long as you can (possibly forever).
- Program for readers, not the computer.
- Don’t be too clever.

Brooks's Laws for Programmers

- Learn from the failures of others.
- A program product is three times as hard to write as a one-off program just for you.
- A system product is three times as hard to write as a program product.
- A project just a little bit bigger than you've ever done before may well be a lot bigger than you can imagine.

Heinlein’s Laws for Writers

- Keep writing until it’s finished.
- When it’s finished, stop writing.
- Keep it on the market until it’s sold.
- When it’s sold, stop marketing.

Combined Laws for Programmers

- Get early feedback.
- Write a simple prototype.
- Rinse and repeat, stop when done.
- Test it!
- Document it!
- Market it!

If it’s any good, in five years you too will be a customer.