Duane Ryan Bailey
23 Haley Street
Williamstown, Massachusetts 01267
(413) 884-2314 - Cell
(413) 551-9348 - Google Voice
bailey.d.r@gmail.com

Duane Ryan Bailey Programming Portfolio

My Interests: My general interests in computing are quite broad. Because I introduced myself to computers at a young age, I've had plenty of time to explore virtually every corner of computer science. Project Euler helped me reason with numbers in a performance-oriented manner, albeit algorithmically instead of constant optimizations. I have also maintained an active interest in projects across the open source spectrum, contributing patches to Python (`http://www.python.org`), to the BeOS revival project of Haiku (`http://www.haiku-os.org`), to LLVM (`http://llvm.org/`), rust (`http://rust-lang.org`), PostgreSQL (`http://www.postgresql.org/`). For every line of code I've ever written, I've read ten, and I've written a lot of code. Some of my favorite projects have been detailed below.

Raytracer: When I was fourteen, I grew interested in Pixar's productions. That computers could produce such realistic images and effects awed me, and I began to research how I might produce similar works. I quickly realized that Pixar's REYES architecture (standing, humorously, for "Renders Everything You Ever Saw") architecture was beyond my grasp, so I turned to raytracing. By the end of the project, I had implemented a substantial renderer, complete with rudimentary global illumination—namely photon mapping and path tracing—textures, and speed enhancements, via k-trees. Though this project has been put aside, I still consider it one of my greatest creations, and certainly the most satisfying for the time invested. Because a ray tracer is both conceptually straightforward and a notoriously difficult problem to optimize, it got me into the world of instruction-level optimization.

FORTH Compiler & OS work: I learned about FORTH when I was fifteen. Though intrigued, I did not really understand it until a year later when I attempted to construct an implementation. The language represented a challenge for me—the language itself represented an entirely new form of programming than I had previously encountered. Furthermore, the process in which the language constructs are defined (named "words") baffled me; branching, for example, happens in a non-intuitive way that took me days to grok. Soon, however, I managed to get simple branching, and a toy FFI. Though not complete, it had all of the core features of FORTH. After a while, I began to wonder about what lay closer to the "metal." What, exactly, the operating system did to make my application work mystified me, and C, along with its `SEGFAULT`s and memory exceptions, was the closest I ever came to understanding the processor. So, to remedy this shortcoming, I set about learning how to write my own operating system. The result was profoundly disappointing—a simple bootstrap usb stick that led to managed memory, simple processes, and a rudimentary prompt. Yet, I learned later that it taught me far more than I realized: an unexpected familiarity with the internal workings of memory and a comfort with the lower level workings of the operating system and, to an extent, the processor itself. This experience also allowed me to implement a bare-bones FORTH implementation in MIPS assembly with an *extremely* simple garbage-collection library on top so that I might understand the architecture better; because I had already experienced the issues I encountered and was able to effectively overcome them.

Scheme Compiler: Sometime after investigating LISP, I decided that I wanted to write my own variation. Because the syntax is, quite literally, a collection of data structures and parsed as thus, the vast majority of language's features can be implemented in the language itself; I only needed to implement a small core—an interpreter and a garbage collector in C to make the language self-sustaining and remain quite fast; it was only a small step to allow it to compile functions, though perhaps slowly, to memory.

I took a hiatus from programming after I left William & Mary to concentrate on my studies. Recently (since my internship with Google), I've gotten back into programming, and these are my recent projects.

tex-view: This is a simple application designed to be a very fast vector document viewer, supporting `DVI` and `PDF` files, with the fast compile-view turnaround of the TEX workflow, automatically reloading from disk.

A modern implementation of TEX: For Spring 2012, I worked on an implementation of TEX in modern C++ as an independent study. This was intended to be a representation of my mastery of proper and efficient memory management, optimization, documentation, testing, and other software engineering skills. However, I have not worked on the project since finishing the independent study, but I want to resurrect this project with techniques I learned at Google and, more recently, research with the PLASMA lab at UMass; the code is still visible at `https://github.com/duane/ctex` but does not reflect what I feel was an immense amount of expertise gained at Google during my internship.

AngularJS: After working at Rap Genius, I decided it was in my best interest to develop my skills with javascript and html. I've always been unhappy with the state of the unholy trinity of HTML/CSS/JS, where I always seemed to be spending my time looking up jQuery calls to manually shuffle elements around the page and keep data in sync. Google's approach of using dependency injection and model-oriented MVC greatly eases testing, DOM interaction, and minimal boilerplate, to the point where I find it actively enjoyable to program for the web and create interactive applications that are more or less consistent from any phone or computer.

Reading: More than anything, however, I spend my time these days reading code. In particular, I have been studying the work of the `PyPy` project, which along with `V8` represents state-of-the-art dynamic compilation, the likes of which haven't been seen outside of lisps and "academic" VMs (namely Self). I've been studying the `llvm` and `clang` projects, and these represent the greatest combination of versatility, speed, and stability of any project I've ever seen before: it is highly readable, allows implementation of virtually any type of language, and allows relatively easy custom compiler instrumentation, representing a breath of fresh air compared to the archaic and opaque `gcc` tools which are, *by design*, difficult to work with. Finally, I've been reading the TeXBook, or the book generated from the literate source code of TEX itself. The combination of the wonderful writing and engineering of Donald Knuth and my love of text and type gives me no end of enjoyment.

Rap Genius: Rap Genius represented my first true experience with web development. They have a stack built from Rails, Redis, and PostgreSQL, and were looking into a mobile application when I joined as an intern. I spent most of my summer immersed in Objective-C, Ruby, and Javascript, and learned a great deal about modern web architectures and, specifically, optimizing complex rendering logic across multiple (ideally parallel) requests and appropriate RESTful practices. Though I was well versed in HTML/CS/JS before this job, it opened my eyes to the great potential of the platform as one of general computing instead of formatted documents.

F1: I worked on F1, a RDBMS, at Google over the summer of 2012, where I partially implemented indexing on (restricted) SQL expressions rather than values directly; for example, this allows case-insensitive string indexing, which is quite useful. I worked in a team environment and gained great experience with source code control, unit and integration testing, code reviews, and working with extremely large code bases. Read the paper at `http://research.google.com/pubs/pub41376.html`.

An "ARP Reflector": William & Mary has many devices (mostly printers and internet-enabled teaching appliances), which upon waking do not advertise their presence; they do, however respond to ARP requests. Unfortunately, the ARP requests would never find the printers if the ARP caches in the various routers and switches around campus have never seen the printer. The solution, which I implemented in C, was to maintain a daemon that listened for incoming ARP requests and "reflected" them to various parts of the network that the requests would usually find unreachable. It was fault-tolerant and maintained a time-variable cache so that these ARP reflectors would not DOS devices on the network from sheer volume of reflected ARP packets, instead only reflecting requests for IPs and that had not been requested in recent history.

A "Smart Logger": Another project I worked on at William & Mary was a simple program that would attempt to parse syslog entries (using `syslog-ng`'s wonderful extensibility) against a series of regular expressions, automatically filing entries into a database. Using this to parse most of the various utilities running on the core W&M servers led to near-instantaneous logging of events across the network, meaning that the system administrators had up-to-date information on every IP, MAC address, switch, server, and program running on the network within seconds of events happening.

A Network Mapping Tool: When I worked at Williams, they primarily hired me to write a program (or aggregation of smaller tools) to map and maintain a network of switches. Because they used both old and new switches, which did not share communication protocols (they had older Cisco switches which supported `SNMP` 1 and 2 and newer HP switches, which only used `SNMP` 3), I had to customize code across a wide variety of switch configurations and operating systems in a fault-tolerant, safe manner, aggregating the data in massive database capable of monitoring switches, tracking devices, and maintaining a (rough) logical map of the network. In retrospect, there are many better ways to acquire the same data, namely running an internal DHCP server that tracks registered computers and communicating with switches upon command or in exceptional situations. Even now, I maintain an active knowledge of switches, networks, and internet theory that has allowed me to approach other, similar projects with a higher degree of confidence than I could have before.

C3D: In Summer 2011 I worked as a contract programmer for the Center for Creative Community Development (`http://www.c-3-d.org/`) at Williams College. In particular, I built a site to view the economic impact of Arts institutions. I parsed Excel spreadsheets using Apache's excellent `POI` framework and evaluated the macros manually to build a system of expressions used to calculate the desired numbers, storing it in XML. Then a PHP fronted loaded the XML to display it on the website, caching results between requests and recalculating the numbers as necessary.

Miscellaneous Projects: I have written numerous, internal patches to open source projects, from the ISC `dhcp` client, `syslog-ng`, `python`, `MongoDB`, among others that I have forgotten; I have proven my ability to immerse myself in foreign code and make sense of the flow of data quickly across a wide variety of languages and programming styles. In addition, I have written countless unix scripts, python modules, and CGI scripts of lesser size. Finally, I have experience constructing Linux distributions meant for PXE booting and Live-CDs, where the systems are meant to boot from read-only hardware and configure themselves automatically to their environment as best as possible.