Programming languages:

* Java
  + Most projects
  + Robotics club
* C#
  + XNA platformer
  + Minecraft clone
  + Unity game engine
* Haskell
  + Programming languages
  + Parsers
  + Computational projects
* Lisp (Racket, Scheme)
  + Basic HTTP server
  + XHTML/XML/SVG generation DSL
  + Interpreter in my operating system’s kernel
* Assembly (x86)
  + Forth implementation
  + Portions of operating system kernel
* Forth
  + Forth implementation
* Rust
  + HTML/XML parser
  + Minecraft server implementation
* Prolog
* C
  + Portions of operating system kernel
  + Virtual machine for my CPU’s ISA
* My programming language, currently unnamed
* (X)HTML(5), CSS, SQL, PHP, XML, JSON
  + Webcomic site for a friend
  + Hierarchical discussion board
  + My personal website

Programming tools:

* Git version control system
  + Branching & merging
  + Semantic versioning
* Visual Studio, Eclipse, Emacs
* Haskell Stack, Cabal
* Continuous integration
  + Jenkins ([https://ci.lijero.co](https://ci.lijero.co/))
  + Travis (<https://travis-ci.org/swittepl/>)
* Unit testing, test-driven development
  + JUnit
  + QuickCheck
* Bugtrackers
  + Github issues
  + Mantis
* Microsoft Office (incl. Word, Excel, PowerPoint)

Systems administration:

* Windows (XP, 7, 8)
* Linux
  + Desktop and server
  + Especially Debian & derivatives
* FreeBSD
  + Used briefly
* Virtual Private Servers (VPSes)
* Servers & daemons
  + Nginx (HTTP & reverse proxy, <https://lijero.co/>)
  + Bind9 (DNS, authoritative for lijero.co)
  + OpenSSH
    - FTP over SSH
  + MariaDB (MySQL)
  + Charybdis (IRC)
* Firewalls
  + iptables
  + Router firewalls
* HTTP stuff
  + Caching
  + Load balancing
  + Content delivery networks (CDNs)
  + Semantic URLs
  + REST
* TLS & certificates
  + ECDHE, RSA
  + Let’s Encrypt
  + DNS Certification Authority Authorization (CAA)
  + OCSP must-staple extension
  + HTTP Strict Transport Security (HSTS)
    - Preloading
  + A+ rating on SSLLabs (<https://www.ssllabs.com/ssltest/analyze.html?d=lijero.co&latest>)
* Operating system internals
  + Kernels (microkernels, monolithic kernels, exokernels)
    - Memory managers
    - Scheduling
      * Scheduling algorithms
      * Priorities
    - Paging
      * Swap space
    - Interrupts, signals, syscalls
  + Filesystems (FAT, others)
  + Network stacks
  + BIOS, UEFI, bootloaders
  + Personally have implemented
    - Basic monolithic kernel (incl. memory manager, paging)
    - Basic filesystem
    - Basic BIOS terminal I/O

Protocols:

* IP
* TCP
  + Could implement
* UDP
* TLS
* HTTP
  + Have implemented, client and server
* HTTP/2
* FTP
  + Could implement
* IRC
  + Have implemented, client and server (partially)
* SMTP
  + Could implement
* Minecraft Protocol (http://wiki.vg/Protocol)
  + Have partially implemented server
  + Not enough for serious gameplay
* Remote procedure calls (RPC)
  + Not a protocol in itself but I understand the concept
* Designed and implemented own protocols
  + Protocol for my own Minecraft clone
    - Used TCP and UDP transport
  + Trivial chat protocol

Former projects:

* Parsers & generators
  + HTML, XML
  + Configuration
  + Command-line arguments
  + Programming language syntax
* Programming language implementations
  + Lisp, Lambda Calculus
    - Too many of these to count
  + Forth
    - In x86 assembly
  + Reactive programming paradigm for Java
    - Via reflection
  + Prolog
    - Unfinished
* Web
  + IRC, HTTP clients and servers
  + Trivial chat program
  + Hierarchical discussion board
  + Personal website
* Low-level
  + Operating system components
    - Monolithic kernel
      * Memory manager
    - Filesystem
    - Terminal
  + CPU in Minecraft with redstone
    - Prior to command blocks or comparators
    - 16-bit, von neuman, stack-based
  + Designed an instruction set and CPU
    - Never physically built
* Games
  + Many, many games in Scratch (scratch.mit.edu)
  + Simple games like Pong, Breakout, Snake, etc
  + Event system
    - In Java, using reflection
  + Component-based object system
    - Also Java
  + Minecraft server implementation
    - Functional, but not usable for gameplay
    - Written in Rust
  + Minecraft clone
    - Written in C# using unity
    - Infinite map generation with simplex noise
    - Mesh generation and optimization
    - Map editing capabilities
    - Networked multiplayer mostly implemented
  + Pipeline and modern OpenGL, shaders
    - Familiarity, but not yet used in practice
  + Minecraft server (60+ concurrent connections at peak)
    - Largely based upon custom content, most of which I wrote myself
      * Included map generation, monster spawns, the monsters themselves, and radiation, temperature, and thirst mechanics
      * This was how I learned to program
    - Custom modpack
      * Self-hosted using a technology called “technic solder”
    - Network of servers via a reverse proxy server called “Bungee”
    - Run on a Linux VPS
    - Taught me to manage large communities
* Robotics
  + Programmer for Robotics club
  + Built most of the internal abstractions
    - Motor controls
    - Input mapping
      * Included dataflow abstraction for input handling
    - Configuration & operation modes

Current Projects:

* Programming language
  + Derived from dependent ordered linear type theory
    - May someday get homotopy theory influences
  + Based on context-sensitive grammar production rules
  + Strong facilities for metaprogramming
  + Multiparadigm
    - Functional
    - Logic programming (backtracking)
    - Concatenative
    - Object-oriented
    - Support for sequential programming and imperative state
    - This can all be implemented via metaprogramming within the language itself
* Rule processing engine
  + Includes internal transactional database
  + Powerful rule-description language
  + Data and code serializability
  + Generate, process, and query data

Suspended projects:

* Operating system
  + “If you use the right programming language, it will all come together”
  + Based on high-level abstractions and languages
  + Not intended to be backwards-compatible
* Overlay network
  + For building distributed or centralized applications
  + Secure, private, and anonymous
  + Datastores or distributed state
  + Can use different models depending upon requirements
    - Blockchain support included

Real-life

Clubs:

* Robotics
* Dungeons and dragons
  + School club
  + Just with friends too
* Band
  + Marching band (football season)
  + Concert band (rest of the year)
  + Jazz band (0-period advanced class before school)

Activities:

* Hiking
  + Especially backpacking
* Cycling
* Skiing
* Dungeons & dragons
* Music
  + Performance and listening
* Board games with friends (& poker with chips)
* Video games, occasionally
  + Dwarf Fortress, Factorio, Terraria with friends
  + Formerly played a lot of Minecraft

Hobbies:

* In addition to those already stated…
* Linguistics & artificial language construction (conlanging)
* Music theory
* Worldbuilding

\* Fundamental knowledge regarding databases: basic SQL, relational algebra, ACID, transactions, CAP theorem, PACELC theorem, CRUD

Theoretical background

* Programming language theory
  + Compiler development
  + Code generation, JIT
  + Interpreters, bytecode
* Type theory
  + Polymorphic, dependent, linear, and non-commutative
* Some category theory
* Formal languages
* Denotational semantics
* Logic
  + Sequent calculus
  + Lambek calculus
  + Formal logic in general
* CPU design and implementations
* Various paradigms
  + Procedural/imperative
    - Loops, pointers, iteration
    - Arrays, strings, characters, integers, floating point, fixed point, booleans, etc
    - Multithreading, mutexes, cooperative multitasking, messages, locks, atomic references, thread pools, reentrancy, thread-local variables
    - Exceptions
    - Records and unions, enums
  + Object-oriented (OOP)
    - Familiar with many design patterns
    - Interfaces, abstract classes, inheritance
    - Generics
    - Reflection
    - Collections
      * Linked lists, array lists, sets, multisets, stacks, trees, graphs, dictionaries (maps), etc
    - Garbage collection
      * Generational
      * Reference counting
    - Encapsulation
      * Various other good principles
  + (Purely) functional
    - Inductive types (product, sum, etc)
    - Coinductive types
    - Continuations, continuation-passing style
      * Coroutines
    - Recursion
    - Lazy vs. eager evaluation
    - Pattern matching
    - Zippers
    - Morphisms
    - Functor, applicative, monoid, monad, traversable, foldable, category, semigroup, etc
    - Arrows, lenses, etc
    - Vertical types, fixpoints, least upper bounds, etc
    - Promises
  + Logic programing, constraint programming
  + Reactive programming