## Programming Skills

### Programming languages

* *Java* – Most of my projects, Robotics Club
* *C#* – XNA platformers, Minecraft clone using Unity game engine
* *C* – Portions of operating system’s kernel, virtual machine for my ISA
* *Rust* – Minecraft server implementation, asset retrieval manager, HTML/XML parser
* *Assembly (x86)* – Forth implementation, portions of my operating system’s kernel
* *Forth* – Forth development environment
* *Haskell* – Programming languages, parsers, computation-heavy projects
* *Lisp (Racket, Scheme)* – HTTP server, XHTML/XML/SVG generation DSL, interpreter in my operating system’s kernel
* *Prolog*
* *Switte* – The temporary name for my work-in-progress programming language (<https://lijero.co/switte>)
* *(X)HTML(5), CSS, SQL, PHP, XML, JSON* – My personal website ([https://lijero.co/)](https://lijero.co/)

### Programming tools

* Git version control system (<https://github.com/lijerom>) – All of my projects (but most of them are *not* on GitHub), semantic versioning
  + Branching & merging – I introduced it to Robotics Club, personal projects
* Visual Studio, Eclipse, Emacs – For C#, Java, and everything else respectively
* Build tools, including
  + Stack (<https://github.com/swittepl/switti>) – For all of my serious Haskell projects
  + Gradle (<https://github.com/lijerom/ab4j/>) – For all of my serious Java projects
* Continuous integration
  + Jenkins ([https://ci.lijero.co](https://ci.lijero.co/))
  + Travis (<https://travis-ci.org/swittepl/>)
* Unit testing, test-driven development (<https://github.com/lijerom/ab4j/>)
  + Junit (<https://ci.lijero.co/job/ab4j/6/testReport/>)
  + QuickCheck
* Bugtrackers
  + Github issues
  + Mantis – Used prior to migrating to Github
* Microsoft Office (incl. Word, Excel, PowerPoint)

## Technical Skills

Servers:

* Servers & daemons
  + *Nginx* (HTTP & reverse proxy) – <https://lijero.co/>
  + *Bind9* (DNS) – The authoritative server for lijero.co
  + *OpenSSH* (SSH, SFTP) – Used to access my servers
  + *MariaDB* (MySQL) – Used by my text board and webcomic sites
  + *Charybdis* (IRC) – Formerly ran one
  + *FileZilla* (FTP) – Formerly ran one
  + *ISS* (HTTP) – Briefly ran one before I had my own server
* HTTP stuff
  + *Caching (client-side)* – used on lijero.co
  + *Caching (server-side)*
  + *Load balancing*
  + *Content delivery networks (CDNs)*
  + *Semantic URLs* – used by my web apps
  + *REST*
* *TLS & certificates*: all of the listed are applicable to my website [https://lijero.co](https://lijero.co/)
  + *ECDHE, RSA*
  + *Let’s Encrypt*
  + *DNS Certification Authority Authorization (CAA)* – motivated me to host my own DNS since my provider didn’t support it
  + *OCSP* – *Must-staple* certificate extension
  + *HTTP Strict Transport Security (HSTS)* – with *Preloading* on all major browsers
  + ***A+ rating on SSLLabs*** (<https://www.ssllabs.com/ssltest/analyze.html?d=lijero.co&latest>)

Systems administration:

* *Virtual Private Servers (VPSes)* – used by my former Minecraft server
* *Firewalls (iptables, router)* – used across my LAN
* *Capability-based security*
* *Disk encryption*

Operating systems:

* *Windows (XP, 7, 8)* – on my desktop
* *Linux, especially Debian* & derivatives – on all of my servers, and my desktop dual-boot
* *FreeBSD* – used briefly
* *Operating system internals* – extensively researched in the pursuit of building my own OS (listed concepts that I have not implemented I could explain in detail if asked)
  + *Kernels* (microkernels, monolithic kernels, exokernels)
    - *Memory managers* – Personally implemented
    - *Scheduling* – SMP, algorithms, priority
    - *Paging* – Personally implemented
      * *Swap space*
    - *Interrupts, signals, syscalls*
  + *Filesystems (FAT, others)* – Personally implemented
  + *Network stacks*
  + *BIOS, UEFI, bootloaders*
  + I personally implemented basic *BIOS terminal I/O*

Protocols:

* *IP*
* *TCP* – Used in many programs and games, learned because I wanted to make a network stack
* *UDP* – Used in my Minecraft clone’s multiplayer
* *TLS/SSL* – Tried and failed to implement the TLS 1.3 spec (<https://tlswg.github.io/tls13-spec/draft-ietf-tls-tls13.html>)
* *HTTP* – Have implemented, client and server, can write requests/responses by memory
* *HTTP/2* – Researched in detail out of interest, serves [https://lijero.co](https://lijero.co/)
* *FTP* – Formerly used often, first protocol I learned, interacted with via telnet
* *IRC* – Have implemented a client and partially a server
* *SMTP* – Have sent emails using telnet
* *Minecraft Protocol* (<http://wiki.vg/Protocol>) – Have partially implemented server
* *Remote procedure calls (RPC)* – Not a protocol in itself but I understand the concept
* Designed and implemented my own protocols
  + *Protocol for my own Minecraft clone*
  + A *trivial chat protocol*

## About Me

I have an insatiable love of learning.

I love the outdoors, including hiking, backpacking, cycling, and skiing.

I am fascinated by knowledge and expression itself: linguistics, programming language theory, formal logic, music theory, computational models, learning algorithms, knowledge representation, and more.

I love creativity in structured contexts such as those described above, including artificial language construction (conlanging), top-down worldbuilding, and Dungeons & Dragons.

I enjoy playing board games and video games with friends.

### Clubs

* *Robotics*
* *Band*
  + *Marching band* – football season
  + *Symphonic band* – rest of the year
  + *Jazz band* – 0-period advanced class before school
* *Dungeons & Dragons* – School club and with friends

and I hope to join *Speech & Debate* next season.

## Experience

2012 – 2015 Java Developer, Owner, Adminstrator – Minecraft Server *“The Mining Dead”*

* 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
* Custom modpack – Self-hosted using a technology called “Technic Solder”
* Network of servers via a reverse proxy called “Bungee”
* *Taught me to manage large communities*
* 60+ concurrent connections at peak

2017 – now Java Developer – Hazen Robotics Club

* I build most of the internal abstractions
  + Motor interfaces
  + Input mapping – Included dataflow abstraction for input handling
  + Configuration & operation modes
* Introduced Git to the team
* Advisor to the team lead

### Self-driven projects

# 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 – Any programming paradigm and syntax can be implemented within the language itself, comparablely to Lisp and Forth macros
* *Rule processing engine*
  + Includes internal transactional database
  + Powerful rule-description language
  + Data and code serializability
  + Generate, process, and query data

# Design-phase / 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

# Former Projects

* *Parsers & generators* – HTML, XML, configuration, command-line arguments, programming language syntax, packets, data serialization, save formats
* *Programming language implementations*
  + *Lisp, Lambda Calculus* – At least one in virtually every language I’ve used
  + *Forth* – Based on JonesForth, x86 assembly
  + *Reactive programming paradigm* for Java via reflection
  + *Prolog* – Unfinished
* Web
  + *Personal website* ([https://lijero.co](https://lijero.co/))
  + *Hierarchical discussion board*
  + *IRC, HTTP* clients and servers
  + *Trivial chat program*
* Low-level
  + *Operating system components*, described in the systems administration section
  + *CPU with Minecraft 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 via reflection
  + *Component-based object system* – in Java
  + *Minecraft server implementation* – in Rust; functional, but not usable for gameplay
  + *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

## Education

2015 – now Hazen High School

### Self-driven learning

In addition to the skills I have listed in other sections, I have learned this theory:

* *Programming language theory* (compiler dev., code generation, JIT, bytecode, interpreters, abstract syntax) – implementing programming languages
* *Type theory* (polymorphic, dependent, linear, and non-commutative) – my programming language’s design
* *Some category theory* – writing abstractions
* *Formal languages* – my programming language’s formal grammar and semantics, linguistics, conlanging
* *Denotational semantics* – used in describing my programming language
* *Logic* (sequent calculus, lambek calculus, formal logic in general) – formal semantics of my programming language
* *CPU design and implementation* – CPU in minecraft, designed a real-life CPU
* *Programming paradigms*
  + *Procedural/imperative* – including threads, exceptions
  + *Object-oriented (OOP)* – includes actually modeling problems, not just using it; abstract classes; generics; reflection; various forms of garbage collection; familiarity with most common collections; and understanding good practices
  + *(Purely) functional* – including, for example, continuations, many common typeclasses, laziness, and common tools like zippers and lenses
  + *Logic programing, constraint programming* – searching a problem domain to find a proof or optimal solution
  + *Reactive programming* – including how to model games, user interfaces, and how it’s implemented
* *Database theory and distributed stores* – including relational algebra, principles such as ACID, and important tradeoffs like the CAP and PACELC theorems
* *Learning algorithms* – Neural networks and evolutionary algorithms