Maynooth University
National University of Ireland Maynooth

Student Name	David P. Callanan	**Student Number**	21444104
Supervisor	Dr. Phil Maguire	**ECTS Credits**	5
Project Title	Studying link-time optimizations in programming language development to facilitate the continuum of static and dynamic modules.		

1 Project Objectives

The goal of this project is to develop a proof-of-concept programming language ("Essence C") that facilitates bespoke optimization techniques via novel language constructs. The specific objective is to research the continuum of static and dynamic modules in low-level software development. A module system will be designed to promote dispatch flexibility by abstracting away the underlying dispatch mechanism, enabling aggressive link-time optimizations when circumstances allow. The development of a prototype compiler will demonstrate the efficacy of the proposed solution.

Success will be evidenced by the following deliverables:

(1) A functional compiler that implements the designed module system;

(2) Documentation of the module system and other core language features;

(3) A simple IDE extension that provides syntax highlighting to the programmer;

(4) A body of research into the continuum of static and dynamic modules in software development, with kernel development as a case study;

(5) An implementation of link-time optimization techniques that leverage the designed module system;

(6) An evaluation of the effectiveness of these optimizations through the benchmarking of sample programs;

(7) A monorepo containing extensive git history, demonstrating ongoing development progress;

(8) A final written report detailing the research, design, implementation and evaluation of the project.

Some items will be out of scope for this project, such as detailed compilation errors, performant compilation and advanced type system features.

2 Description of Work Completed

Preliminary work on syntax design and language parsing code has been completed, and background research into the LLVM toolchain has been conducted. AI assistance was also briefly assessed.

2.1 Evidence of Work Completed

Initially, a handful of pseudocode files were created to brainstorm syntax ideas for the language. Then, a simple VS Code extension was developed to provide syntax highlighting for these early syntax ideas to reduce eye strain. Next, work commenced on the compiler frontend, written in JavaScript. Some previously written code from personal projects was used (including a simple parsing library), justifying the decision to use JavaScript. Basic parse-tree generation was implemented for a portion of the preliminary syntax, and a custom "source scope" system was developed to handle the translation between filesystem resources and source references within the programming language. Next, the build system for the compiler backend (written in C++) was arranged. A Makefile was used within a Dockerized environment, itself orchestrated by a Linux/WSL environment for best reproducibility. C++ was justified due to its first-class integration with the LLVM toolchain [1].

2.2 Literature Review

LLVM was found to be a practical compilation target for low-level programming languages due to (1) its target-independent code generation [2] and (2) its integration capabilities with existing programming languages thanks to ABI interoperability (including some recent improvements) [3]. Furthermore, the extensive optimization passes shared amongst languages targetting LLVM [4] should make *Essence C* directly comparable with existing languages, allowing benchmarks to be solely influenced by the additional optimization techniques employed. For instance, C can be compiled with LLVM via the popular Clang compiler [5]. It was also noticed that LLVM has experimental support for inline assembly [6], preventing the layer of indirection of a separate compilation unit (such as in kernel development [7]). Additionally, LLVM benefits greatly from intermodular optimizations at link-time [8], because its intermediate representation (IR) preserves the necessary higher-level constructs. Critically, the LLVM API exposes a powerful interface to programatically manipulate existing LLVM IR files [9] [10]. It follows that inlined compatibility wrappers can be generated to avoid optimization barriers between compilation units originating from languages other than *Essence C*.

2.3 Use of GenAI and Tools

Two notable GenAI tools have been used to assist with the research and development of this project so far.

(1) The first is t3.chat, a web application that consolidates numerous large language models into a streamlined chat interface [11]. The user can experiment with different models to understand which models are best suited for the task at hand. The tool was primarily used to steer the work in the right direction and to identify the concepts, libraries and resources through which further research could be conducted. For transparency, all LLM prompts have been included in **Appendix B**.

(2) The second is the tab-complete feature of GitHub Copilot. This tool has encouraged rapid development and iteration through its behaving as a glorified autocomplete. From experience, the tool's context is severely limited, forcing the developer to have genuine knowledge of the codebase and its architecture. This is beneficial for projects that require significant unique input from the developer, in contrast to rehash projects that benefit the most from LLM assistance.

Some experimentation of GitHub Copilot agent mode is ongoing, with notable issues identified. Firstly, the development speed is impeded by slow model response times, lack of funds, and internet connectivity issues. Secondly, there is a great difficulty in referencing contributions made by this tool due to (1) tool calls that modify the codebase directly and (2) significant implicit pollution of the context window as the agent navigates the respository. As such, the usage of this tool has been temporarily suspended pending further assessment.

3 Future Work

The core of the project revolves around the design of the module system. The accompanying research will be a significant undertaking, including static vs dynamic dispatch, instance tracking logic, and the feasibility of IR interoperability in the proposed optimization techniques. This will be tackled first, in parallel with the development of any necessary boilerplate code for the compiler backend. Some experimentation will be needed with LLVM IR generation, before starting on the final compiler. Everything except the final compiler is expected to be completed by the end of January, leaving a significant amount of time for implementation, testing and evaluation of performance.

References

[1] "Frequently Asked Questions (FAQ) — LLVM 22.0.0git documentation". https://llvm.org/docs/FAQ.html#in-what-language-is-llvm-written

[2] "The LLVM Target-Independent Code Generator — LLVM 22.0.0git documentation". https://llvm.org/docs/CodeGenerator.html

[3] "GSoC 2025: Introducing an ABI Lowering Library - The LLVM Project Blog". https://blog.llvm.org/posts/2025-08-25-abi-library/

[4] "(Inline Assembler Expressions) LLVM's Analysis and Transform Passes — LLVM 22.0.0git documentation". https://llvm.org/docs/Passes.html#domtree-dominator-tree-construction

[5] "Clang C Language Family Frontend for LLVM". https://clang.llvm.org/

[6] "LLVM Language Reference Manual — LLVM 22.0.0git documentation". https://llvm.org/docs/LangRef.html#inline-assembler-expressions

[7] "GCC Inline Assembly and Its Usage in the Linux Kernel". https://dl.acm.org/doi/fullHtml/10.5555/3024956.3024958

[8] "LLVM Link Time Optimization: Design and Implementation — LLVM 22.0.0git documentation". https://llvm.org/docs/LinkTimeOptimization.html

[9] "LLVM Programmer's Manual — LLVM 22.0.0git documentation". https://llvm.org/docs/ProgrammersManual.html

[10] "Writing an LLVM Pass — LLVM 22.0.0git documentation". https://llvm.org/docs/WritingAnLLVMNewPMPass.html

[11] "Meet T3 Chat AI: Your All-in-One, Super-Fast AI Assistant," DigiVirus. https://digivirus.in/meet-t3-chat-ai-your-all-in-one-super-fast-ai-assistant/

A Appendix: Source Code

Included in this appendix is a snapshot of the project repository as of the date of submission of this interim report. The full repository, including the entire git history, is available upon request. Please request access to the private repository located at https://github.com/davidcallanan/mu-fyp.

Build artifacts and other uninteresting files have been omitted from this appendix. This includes, but is not limited to, all files mentioned inside .gitignore.

.gitmodules

```
[submodule "compiler/frontend/uoe"]
    path = compiler/frontend/uoe
    url = https://github.com/davidcallanan/uoe
```

2510_syntax_ideas/idea01.ec

```
what i want to do is write my own language, but allow to user to link
   ↪ in with llvm ir should any functionality not be feasible in my
   ↪ language (e.g. specific optimization flags etc).

sources :intf {
    /x/y/z /blah/wah/frah
}

sources :impl {
    /x/y/z /blah/wah/frah
}

deps {
    :print (/print, :static(/x86_64/print, :object_wide:global)); //
   ↪ uses one inter-object implementation
    :print (/print, :static(/x86_64/print, :object_wide:named(/
   ↪ my_lovely_instance)));
    :print (/print, :static(/x86_64/print, :local)); // any instance of
   ↪  this module has its own implementation
    :print (/print :)
}

no because i think the implementation of being static etc actually
   ↪ depends on the bigger picture

deps {
    :print (/print, :concrete(/x86_64/print, :local, :static));
```

```
    :print (/print, :concrete(/x86_64/print, :named(/my_implementation)
   ↪ , :static));
    :print (/print, :passed:named(/my_implementation));
}

# this is tough

maybe a distinction for each "module" of what's "internal" to that
   ↪ module and what is "external". anything internal it is the
   ↪ responsibility of the module author to instantiate one with
   ↪ internal aspects prepopulated.

in another module, loading up a module it is assumed that anything
   ↪ internal is pre-populated.

when something is internal it is understood that you cannot create "
   ↪ multiple instances" (not of the module but of the module factory
   ↪ i mean). thus, that module can be configured at a global level.

instantiate modsource ();

:x86_64:print

do i really need a / variant?

print := import_intf:print;

the idea is that / is subject to translation.

any functions that work on the same "realm" will automatically use the
   ↪ designated translator when taking input. [for now, this
   ↪ translator is not going to exist, but the point is it could
   ↪ eventually].
```

2510_syntax_ideas/idea02.ec

```
; this is a comment because why not! i think it would be pretty cool.
; double slash means go to parent root
; triple slash means go to parent root and find its parent root
; there is a cap on triple slash. if you need to go further, you must
   ↪ then start doing mappings within the parent root "sources" entry.
; this ensures we dont go absolutely crazy.

reset-double-root; this resets // to current module
reset-triple-root; this resets /// to current module
```

```
sources {
    /// / ; maybe this is a better way to reset?
    // /
    /// reset; or reset like this
    /deps/foo //deps/foo
    /deps/bar ///deps/bar
    print("hello");
    ; only sources can access // and ///, using this anywhere else
   ↪ would be illegal, so you have to create a local mapping.
    // reset; must come at the end because previous mappings will refer
   ↪  to old version... or not, the user can just have the common
   ↪ sense to know that this only impacts child modules.
    /// resetchildren; this resets only for children.
    ; or preferably
    double-reset;
    triple-reset;
    triple-reset-children;
    double-reset-children;
    ; or maybe
    reset-double;
    reset-triple;
    reset-triple-children;
    reset-double-children;
}
```

2510_syntax_ideas/idea03.ec

```
sources {
    /intf/foo //intf/foo;
    /intf/bar //intf/bar;
    ; if reset /// appears here, it wouldn't be accessible in
   ↪ forwarding blocks
}

forwarded sources {
    reset ///;
    ; if reset /// appears here, it wouldn't be forwarded but would
   ↪ still be accessible in forwarding blocks
}

public namespace foo = import /intf/foo;

namespace bar = import /intf/bar {
    reset //;
    //intf/foo /intf/foo
    //banana/apple //banana/apple
```

```
    //banana/grape ///banana/grape
}

type Foo = foo::Foo;

public type Bar = bar::Bar;

; should anything in triple be automatically available in double?
; maybe the other way around? if we access triple, it will fallback to
   ↪ double?
; no it doesn't do justice.
```

2510_syntax_ideas/idea04.ec

```
sources {
    /intf/foo //intf/foo;
    /intf/bar //intf/bar;
    /intf/baz //intf/com.dcallanan/baz;
    ; if reset // appears here, it wouldn't be accessible in forwarding
   ↪  blocks
}

forwarded sources {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
   ↪ still be accessible in forwarding blocks
}

public namespace foo import /intf/foo;

namespace foo_bar import /intf/bar {
    reset //;
    //intf/foo /intf/foo;
    //banana/apple //banana/apple;
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Foo map {
    :vfs map (Foo);
    :blah map (*Bar);
    :wah Foo; syntactic sugar for :wah map (Foo)
}
```

```
type Bar enum {
    :my_entry_name;
    :my_other_entry_name (

    );
}

type DeliciousInteger i33;
```

2510_syntax_ideas/idea05.ec

```
sources {
    /intf/foo //intf/foo;
    /intf/bar //intf/bar;
    /intf/baz //intf/com.dcallanan/baz;
    ; if reset // appears here, it wouldn't be accessible in forwarding
    ↪  blocks
}

forwarded sources {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
    ↪ still be accessible in forwarding blocks
}

public namespace foo import /intf/foo;

namespace foo_bar import /intf/bar {
    reset //;
    //intf/foo /intf/foo;
    //banana/apple //banana/apple;
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Bar enum {
    :my_entry_name;
    :my_other_entry_name map (
        :vfs Foo;
        :bar i32;
    );
}

type Foo map {
```

```
    :vfs Foo; syntactic sugar for :vfs map (: Foo);
    :blah *Bar;
    :point map (i32, i32, i32);
    :distance(:x i32, :y i32) i32;
    (:x i32, :y i32) f64;
    : Foo;
}

type DeliciousInteger i33;
```

2510_syntax_ideas/idea06.ec

```
sources {
    /intf/foo //intf/foo;
    /intf/bar //intf/bar;
    /intf/baz //intf/com.dcallanan/baz {
        ; allows controlling with higher priority what is populated
    ↪ when this is later used
        ; for example if this is later passed down to other modules,
        ; it will base off the "//" at this scope,
        ; not what "//" later happens to be.
        /blah/wah //blah/wah;
        ; / always means the "module root" even though we could be
    ↪ touching a directory that does not reset the module root, recall
    ↪ the difference
        ; between the raw filesystem and the module hierarchy.
        ; thus / is the current module root irrespective of the
    ↪ directory or file mentioned.
        ; and // is the current "outer root" that this module is
    ↪ operating under.
    }
    ; if reset // appears here, it wouldn't be accessible in forwarding
    ↪  blocks
}

forwarded sources {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
    ↪ still be accessible in forwarding blocks
}

public namespace foo import /intf/foo;

namespace foo_bar import /intf/bar {
    reset //;
    //intf/foo /intf/foo;
```

```
    //banana/apple //banana/apple;
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Bar enum {
    :my_entry_name;
    :my_other_entry_name map (
        :vfs Foo;
        :bar i32;
    );
}

type Foo map {
    :vfs Foo; syntactic sugar for :vfs map (: Foo);
    :blah *Bar;
    :point map (i32, i32, i32);
    :distance(:x i32, :y i32) i32;
    (:x i32, :y i32) f64;
    : Foo;
}

type DeliciousInteger i33;
```

2510_syntax_ideas/idea07.ec

```
sources {
    /intf/foo //intf/foo;
    /intf/bar //intf/bar;
    /intf/baz //intf/com.dcallanan/baz {
        ; allows controlling with higher priority what is populated
    ↪ when this is later used
        ; for example if this is later passed down to other modules,
        ; it will base off the "//" at this scope,
        ; not what "//" later happens to be.
        /blah/wah //blah/wah;
        ; / always means the "module root" even though we could be
    ↪ touching a directory that does not reset the module root, recall
    ↪ the difference
        ; between the raw filesystem and the module hierarchy.
        ; thus / is the current module root irrespective of the
    ↪ directory or file mentioned.
        ; and // is the current "outer root" that this module is
    ↪ operating under.
```

```
    }
    ; if reset // appears here, it wouldn't be accessible in forwarding
   ↪  blocks
}

forwarded sources {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
   ↪ still be accessible in forwarding blocks
}

public namespace foo import /intf/foo;

namespace foo_bar import /intf/bar {
    reset //;
    //intf/foo /intf/foo;
    //banana/apple //banana/apple;
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Bar enum {
    :my_entry_name;
    :my_other_entry_name map (
        :vfs Foo;
        :bar i32;
    );
}

type Foo map {
    :vfs Foo; syntactic sugar for :vfs map (: Foo);
    :blah *Bar;
    :point map (i32, i32, i32);
    :distance(:x i32, :y i32) i32;
    (:x i32, :y i32) f64;
    : Foo;
}

type DeliciousInteger i33;
```

2510_syntax_ideas/idea08.ec

```
sources {
    /intf/foo //intf/foo;
```

```
    /intf/bar //intf/bar;
    /intf/baz //intf/com.dcallanan/baz {
        ; allows controlling with higher priority what is populated
    ↪ when this is later used
        ; for example if this is later passed down to other modules,
        ; it will base off the "//" at this scope,
        ; not what "//" later happens to be.
        /blah/wah //blah/wah;
        ; / always means the "module root" even though we could be
    ↪ touching a directory that does not reset the module root, recall
    ↪ the difference
        ; between the raw filesystem and the module hierarchy.
        ; thus / is the current module root irrespective of the
    ↪ directory or file mentioned.
        ; and // is the current "outer root" that this module is
    ↪ operating under.
    }
    ; if reset // appears here, it wouldn't be accessible in forwarding
    ↪  blocks
}

forwarded sources {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
    ↪ still be accessible in forwarding blocks
}

public namespace foo import /intf/foo;

namespace foo_bar import /intf/bar {
    reset //;
    //intf/foo /intf/foo;
    //banana/apple //banana/apple;
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Bar enum {
    :my_entry_name;
    :my_other_entry_name map (
        :vfs Foo;
        :bar i32;
    );
}
```

```
type Foo map {
    :vfs Foo; syntactic sugar for :vfs map (: Foo);
    :blah *Bar;
    :point map (i32, i32, i32);
    :distance map map (:x i32, :y i32) i32;
    :distance map (:x i32, :y i32) -> i32; i think this looks better
    map map (:x i32, :y i32) f64;
    map (:x i32, :y i32) -> f64; i think this is good stuff
    : Foo;
}

type DeliciousInteger i33;
```

2511_syntax_ideas/idea01.ec

```
sources {
    ; probably will scrap a regular sources block because we already
   ↪ have a layer of naming indirection when "importing" should that
   ↪ not be sufficient?
    ; each import then involves a concious choice of using either "/" (
   ↪ i own the module!) or "//" (i am using something external). it
   ↪ doesn't require much effort to change between them if you change
   ↪ your mind.
}

forwarding {
    reset //;
    ; if reset // appears here, it wouldn't be forwarded but would
   ↪ still be accessible in forwarding blocks for individual imports
   ↪ to override if so wished.
    ; the default strategy is not to reset, so that forwarding is
   ↪ automatic.
    ; LHS can only be // here, but RHS can be / or //.
    ; resets must be the very first thing if wanted, otherwise no reset
   ↪ .
    ; every module implicitely resets /.
}

; the key is that a module always owns its types when it comes to the
   ↪ type system (irrespective of whether it owns the codebase when it
   ↪  comes to importing).
; thus it seems rather convenient to be able to export a namespace.
public types foo import /intf/foo;

types foo import /intf/foo;
```

```
types foo_bar import /intf/bar forwarding {
    reset //;
    //intf/foo /intf/foo;
    //banana/apple //banana/apple;
    ; like other forwarding blocks, LHS must be //, but RHS can be / or
   ↪  //.
}

type Foo foo_bar::FooBarf;

public type Bar foo_bar::Bar;

type Bar enum {
    :my_entry_name;
    :my_other_entry_name map (
        :vfs Foo;
        :bar i32;
    );
}

type Foo map {
    :vfs Foo; syntactic sugar for :vfs map (: Foo);
    :blah *Bar;
    :point map (i32, i32, i32);
    :distance map map (:x i32, :y i32) i32; i don't want to use this
   ↪ syntax
    :distance map (:x i32, :y i32) -> i32; i think this looks better
    ; the idea is that "->" automatically means a map that takes in non
   ↪ -sym input
    ; of course then in the following line we make the current map
   ↪ itself callable with non-sym.
    ; internally it is taught of as a sort of ":call" symbol, because i
   ↪  like making anything callable.
    map (:x i32, :y i32) -> f64; i think this is good stuff
    : Foo; this is what the type is when not treated as a map. the "
   ↪ leaf value" as i have called it for many years.
}

type DeliciousInteger i33;

; now in source modules, we need to think how we take in dependencies
   ↪ and all that and instantiate a module
; before i think i was thinking that the source module specifies things
   ↪  like how it wants to grab these dependencies (static vs dynamic)
   ↪ , but i think the module user should be able to control that.
; really dependencies is just a map?
```

```
dependencies map {
    :foo Foo; we want an instance of foo
    :bar_factory BarFactory;
}

options map {}

; however i don't want to create special language constructs for this.
   ↪ i wonder is it possible to make module instantiating a regular
   ↪ function. well, it is indeed a comptime function, or is it.
; no it's not, it can be runtime, so we can pass in dependencies.
; but at some stage we can specify that a variable is "static" or
   ↪ something like this, and the compiler has to trace that and
   ↪ remember it.
; the tracing might be difficult.

; i guess first, we should look at how do we import a source module
   ↪ compared to intf, and probably similar.

types foo import /impl/foo;

; i guess `foo` now has some types in it but then there would also be a
   ↪  way to instantiate it.
; there could be a cool thing i mention in my report "compile-time
   ↪ tracing of the static vs dynamic nature through runtime code".

; maybe we just take stuff "in".

in map {
    :dependencies map {
        :foo Foo; we want an instance of foo
        :bar_factory BarFactory;
    }

    :options map {
        :lru_size i32;
    }
};

; one can repeat "in map" blocks and they get merged.
; we might go ahead and force "in" map to be a non-tuple map to enable
   ↪ this straightforward merge behaviour.

; any instance of a module keeps track of whether it is a runtime or
   ↪ comptime instance.
; some simple rules can then be applied:
; when we create an instance of a module, and pass it a dependency, if
```

```
    ↪ the dependency is clearly comptime (say at the global scope), the
    ↪  instance remembers this.
; if runtime then also remembers this.
; but the key is that comptime can be reduced to runtime (if any chance
    ↪  of runtime) but runtime can not ever go back up to comptime.
; we can enforce comp-time perhaps through a little keyword (and simply
    ↪  the compiler would choose to bail out if it can't prove it).
; by default when you create an instance of a module, it is made as
    ↪ comptime if the instance itself is comptime (and maybe instead of
    ↪  comptime we are thinking of the word "static").

; im trying to think it through. i guess the root "entry" module we
    ↪ compile ought to be comptime.
; if we access something straight from "input" then this comptime is
    ↪ preserved.
; but let's say we call some function or something to obtain the
    ↪ instance, then comptime is gone.
; wait, but what about when we genuinely instantiate a module instance
    ↪ .... ahhh

; i think it comes back to, do I own the instance or not? (similar to
    ↪ do i own the code?)

; let's suppose we instantiate a module instance...
; i guess a namespace can have attached to it an implementation,
    ↪ allowing instantiation.
; it's like the idea of leaf functions, but here, we are thinking to
    ↪ ourselves that there are no leaf functions anymore, just the one
    ↪ "instantiation" function. further non-leaf instantiation
    ↪ functions can always be created if so desired.

my_module := instantiate foo(
    :dependencies {
        ; blah
    }
    :options {
        ; blah
    }
);

; notice that this instantiation is done at the global scope, thus it
    ↪ can inherit the comptime of the current module instance.
; while properties in maps could also keep track of comptime vs runtime
    ↪ , i feel this might be too complex for a fyp that is due in March
    ↪ .
; so let's just simplify for now and so VARIABLES at module scope keep
    ↪ track of comptime vs runtime.
```

```
; variables are easier to deal with.. don't got the complexities of a
   ↪ map with types and all that.

; note though: the "input" is a map, so we do have to keep track
   ↪ through maps ahhhhh

; right how via gonna track via maps? remember we have a few
   ↪ possibilities, we have static, we have thread-local storage or
   ↪ global variable, and we have dynamic.
; however, we can assume tls and gv are static, as a static
   ↪ implementation can wrap the dynamic nature.
; we can use LTO (link time optimization) to easily do this (which
   ↪ would be great to talk about in my report).

; in my report: "technically C does not have to even compile to object
   ↪ files, it could compile each file to an executable file and then
   ↪ use an overkill runtime linkage mechanism, the point is that
   ↪ there is room for alteration in this".

; so all "sym" map instances (don't care about non-sym variants) must
   ↪ keep track of their comptime vs runtime status, collapsing to the
   ↪  worst-case scenario (kind of like how a typesystem collapses its
   ↪  possible values).

; by default, dispatch is "static", but there would be perhaps three
   ↪ options for this demo.

my_module := instantiate foo(

) dispatch {
    static force
};

my_module := instantiate foo(

) dispatch {
    global
};

my_module := instantiate foo(

) dispatch {
    dynamic
};

; force will ensure that the compiler bails out if not possible, but by
   ↪  default, it is just a recommendation.
```

```
; when one instantiates a module in another scope, then dispatch is
   ↪ ignored and it is just treated as dynamic, like instantiating a
   ↪ class.

; however, what if we instantiate two modules that both implement that
   ↪ interface and might pass through?
; i think when we say dynamic, we do in fact mean double dispatch,
   ↪ unfortunately. there is an element of saying the implementation
   ↪ is fixed, but its dependencies etc. is dynamic, so maybe we can
   ↪ consider dyanamic-single vs dynamic-double.
; even some of these things can be put through as considerations in my
   ↪ report, whether i implement or not (because implementing dynamic
   ↪ gonna be hell of a lot easier).

; i think i can make effort to propogate dyanmic-single vs dynamic-
   ↪ double,,, we'll seee..



; im thinking when importing a source module it will instaed be

mod my_module import /impl/my_module;

; we distinguish between "mod" and "ns" because "mod" has instantiation
   ↪  function.
```

2511_syntax_ideas/idea02.ec

```
; i was leaning towards not having special treatment for module
   ↪ instances and that we can have a receiver-like system for any
   ↪ sort of instances.
; but we would force that modules require an instance, in the sense
   ↪ that we have leaf maps that can't do much by themselves.

; it just could get messy if a module has the ability to create a
   ↪ vector etc.
; i might go with implicit module receivers (for anything called
   ↪ create_mod)?
; so instead of this:mod:blah we'd just have mod:blah.
; or maybe module gets special treatement i don't know at this point.

create(:my_dependency Foo, :math Math) {
    this:my_dependency @= my_dependency; // @= says "declare and assign
   ↪ if not already declared", whereas ":= is for variables and can
```

```
    ↪ have shadowing, whereas @= we know like there's genuinely only one
    ↪  version".
    this:math @= math;
}

@:create_world() {
    mod:my_dependency:do_something;
}

@:create_world:create_vector(x f64, y f64, z f64) {
    mod:my_dependency:do_something_or_another;

    this:x @= x;
    this:y @= y;
    this:z @= z;
}

@:create_world:create_vector:get_x() -> f64 {
    return this:x;
}

@:create_world:create_vector:get_y() -> f64 {
    return this:y;
}

@:create_world:create_vector:get_z() -> f64 {
    return this:z;
}

@:create_world:create_vector:get_distance(other T:create_world:
    ↪ create_vector) -> f64 {
    return mod:math:sqrt(
        + (this:x - other:get_x) * (this:x - other:get_x)
        + (this:y - other:get_y) * (this:y - other:get_y)
        + (this:z - other:get_z) * (this:z - other:get_z)
    );
}

; i can talk about how forcing no ability for global variables forces
    ↪ the compiler to track everything. preparing us nicely for
    ↪ optimizations.

; need some self-loop of wanting to actually get the exposed version of
    ↪  "this", like "this~" or something.

; ah what now about private functions within my shtuff! now i need a
    ↪ symbol for private too?
```

```
; also look now carefully how awful the typing syntax is...
; maybe i do need to have type for returned instances, but damn i didn'
   ↪ t want this.
; also note that i want want different factories for a type, but really
   ↪  the other factories should wrap a main internal factory kind of.

; also create_vector doesn't know any context about create_world unless
   ↪  the information of world is stored in vector's this... so it's
   ↪ kinda like not really appropriate to even have this create_world
   ↪ mentioned...

; note also that maybe if we do have like a Vector declaration that we
   ↪ need only include mutable things in here (to prevent the need for
   ↪  @=) and anything constant can be extended in the same way methods
   ↪  are.
```

2511_syntax_ideas/idea03.ec

```
; let's have another go at idea02.

type Mod map {
    :_my_dependency Foo;
    :_math Math;
}

create(:my_dependency Foo, :math Math) {
    ; if constructor finds something not assigned, compile-time error.
    mod:_my_dependency = my_dependency;
    mod:_math = math;
    ; i'm thinking of _ here but in practice i think i want to follow
   ↪ the public approach where there is no such thing as module-level
   ↪ private.
}

type World map {
}

; the idea is that one should be able to immediately start doing stuff
   ↪ with @Mod without even using "create" or "type Mod map" unless
   ↪ they actually need it.

@Mod:create_world() -> World { ; we want () here to allow later
   ↪ extensibility.. this is just a convention that programmers would
   ↪ likely follow.
    mod:_my_dependency:do_something;
```

```
}

type Vector map {
    :_x f64;
    :_y f64;
    :_z f64;
}

@World:create_vector(x f64, y f64, z f64) -> Vector {
    mod:_my_dependency:do_something_or_another;

    this:_x = x;
    this:_y = y;
    this:_z = z;
}

@Vector:get_x() -> f64 {
    return this:_x;
}

@Vector:get_y() -> f64 {
    return this:_y;
}

@Vector:get_z() -> f64 {
    return this:_z;
}

@Vector:get_distance(other T:Vector) -> f64 {
    return mod:_math:sqrt(
        + (this:_x - other:get_x) * (this:_x - other:get_x)
        + (this:_y - other:get_y) * (this:_y - other:get_y)
        + (this:_z - other:get_z) * (this:_z - other:get_z)
    );
}
```

2511_syntax_ideas/idea04.ec

```
; the goal of this iteration is to determine how public private should
   ↪ work
; and to resolve the notations of maps (RHS) of @ extensions.
; i'd also like to determine how to make things mutable.

; anything mutable must be defined in the type. how about we suppose
   ↪ that this is not necessary for sake of exploration.
```

```
@Mod:my_dependency Foo;
@Mod:math Math;

create(:my_dependency Foo, :math Math) {
    mod:my_dependency = my_dependency;
    mod:math = math;
}

type World map;

pub @Mod:create_world map () -> World . { ; . means implicit? this
    ↪ distinguishes between type details and actual body.
    mod:my_dependency:do_something;
}

type Vector map;

mut @Vector:x f64; mut kinda means non-deterministic
mut @Vector:y f64; these should mutable because maybe we have a method
    ↪ to mutate them
mut @Vector:z f64;

pub @World:create_vector map (x f64, y f64, z f64) -> Vector . {
    mod:my_dependency:do_something_or_another;

    ; this is problematic. this could be referring to world, or it
    ↪ could be referring to vector instance.

    this:x = x;
    this:y = y;
    this:z = z;
}

pub @Vector:get_x map . {
    return this:x;
}

pub @Vector:get_y map . {
    return this:y;
}

pub @Vector:get_z map . {
    return this:z;
}

pub @Vector:get_distance map (other Vector) -> f64 . {
    return mod:math:sqrt(
```

```
        + (this:x - other:get_x) * (this:x - other:get_x)
        + (this:y - other:get_y) * (this:y - other:get_y)
        + (this:z - other:get_z) * (this:z - other:get_z)
    );
}

; on the next iteration i need to resolve some things:
; - this does not work in factories very nicely... ahhh!!!
; - . for implicit is weird... i need to think that through more.
```

2511_syntax_ideas/idea05.ec

```
; i think we already decided prior that the whole point of prefixing
    ↪ maps with "map" at the type level is so that we can take
    ↪ advantage of () and {} notation for instances.

; so i wonder can we get going with that immediately.

@Mod:my_dependency Foo;
@Mod:math Math;

create(:my_dependency Foo, :math Math) {
    :my_dependency my_dependency;
    :math math;
}

type World map;

pub @Mod:create_world() -> World { ; here World is kinda optional, and
    ↪ we can make everything optional because we know the difference
    ↪ between type and instance at the lexer level almost.
    mod:my_dependency:do_something;
}

type Vector map;

mut @Vector:x f64;
mut @Vector:y f64;
mut @Vector:z f64;

pub @World:create_vector(:x f64, :y f64, :z f64) -> Vector {
    mod:my_dependency:do_something_or_another;

    ; normally we would return things like this:

    :x x;
```

```
    :y y;
    :z z;

    ; but remember! these are private!
    ; seems dodgy.
    ; then again, the notion of private is a module-level thing (Go ftw
    ↪ ).
    ; there's not supposed to be the ability to do private.
    ; if you need truly private and temporary, then use a local
    ↪ variable
    ; otherwise you just gotta use a field.
    ; you can internally decide on a _x convention if wanted, but my
    ↪ thinking is that this is unnecessary
    ; since i myself have accessed _x things outside where i thought i
    ↪ would in javascript many times, that's the whole point of having
    ↪ a module level of privacy.
}

pub @Vector:get_x f64 {
    return this:x;
}

pub @Vector:get_y f64 {
    return this:y;
}

pub @Vector:get_z f64 {
    return this:z;
}

pub @Vector:get_distance(other Vector) -> f64 {
    return mod:math:sqrt(
        + (this:x - other:get_x) * (this:x - other:get_x)
        + (this:y - other:get_y) * (this:y - other:get_y)
        + (this:z - other:get_z) * (this:z - other:get_z)
    );
}

; this actually looks quite good.
```

2511_syntax_ideas/idea06.ec

```
@Mod:my_dependency Foo;
@Mod:math Math;

create(:my_dependency Foo, :math Math) {
```

```
    :my_dependency my_dependency;
    :math math;
}

type World map;

pub @Mod:create_world() -> World {
    mod:my_dependency:do_something;
}

type Vector map;

mut @Vector:x f64;
mut @Vector:y f64;
mut @Vector:z f64;

pub @World:create_vector(:x f64, :y f64, :z f64) -> Vector {
    mod:my_dependency:do_something_or_another;

    :x x;
    :y y;
    :z z;
}

pub @Vector:get_x f64 {
    return this:x;
}

pub @Vector:get_y f64 {
    return this:y;
}

pub @Vector:get_z f64 {
    return this:z;
}

pub @Vector:get_distance(other Vector) -> f64 {
    return mod:math:sqrt(
        + (this:x - other:get_x) * (this:x - other:get_x)
        + (this:y - other:get_y) * (this:y - other:get_y)
        + (this:z - other:get_z) * (this:z - other:get_z)
    );
}
```

2511_syntax_ideas/idea07.ec

```
types foo import //intf/foo;

mod bar import /impl/bar forwarding {
    reset //; todo: i need to update my code to allow resetting any "
    ↪ folder"?
    reset //foo/bar/baz;
    //intf/foo //intf/foo;
}

type Foo foo::Foo;
type Math bar::Math;

@Mod:my_dependency Foo;
@Mod:math Math;

create(:my_dependency Foo, :math Math) -> {
    :my_dependency my_dependency;
    :math math;
}

type World map;

pub @Mod:create_world() -> World {
    mod:my_dependency:do_something;
}

type Vector map;

mut @Vector:x f64;
mut @Vector:y f64;
mut @Vector:z f64;

pub @World:create_vector(:x f64, :y f64, :z f64) -> Vector {
    mod:my_dependency:do_something_or_another;

    :x x;
    :y y;
    :z z;
}

pub @Vector:get_x f64 {
    return this:x;
}

pub @Vector:get_y f64 {
    return this:y;
}
```

```
pub @Vector:get_z f64 {
    return this:z;
}

pub @Vector:get_distance(other Vector) -> f64 {
    first_part := (this:x - other:get_x) * (this:x - other:get_x)
    second_part := (this:y - other:get_y) * (this:y - other:get_y)
    third_part := (this:z - other:get_z) * (this:z - other:get_z)

    : mod:math:sqrt(
        + first_part
        + second_part
        + third_part
    );
}
```

README.md

```
# Essence C (Maynooth University - Final Year Project)

This respository acts as the definitive record of work for my final
   ↪ year project. I will try to keep as much of my work as possible
   ↪ within this repo - including code, writeups, notes, random
   ↪ thoughts.

The goal of this project is to develop a custom programming language
   ↪ known as "Essence C".

The language should include bespoke mechanism(s) to facilitate a
   ↪ flexible module dispatch mechanism - primarily achieved by fusing
   ↪  the notions of classes and modules that are traditionally
   ↪ separate in existing programming languages.

The end goal of this is to enhance the compiler's potential to perform
   ↪ link-time optimization on the resulting artifacts, resulting in
   ↪ small performance optimizations relative to other low-level
   ↪ languages (if things work out smoothly).

Additional research (if time permits) will be to study how this
   ↪ facilitates the continuum of both static and dynamic modules in
   ↪ kernel development. This may (if time permits) involve writing
   ↪ some simple kernel code in my programming language and analyzing
   ↪ performance gains relative to C code. The goal will be to get the
   ↪  project working before focusing on these aspects though.
```

```
One core aspect of the project will be attempting to maintain
    ↪ interconnectivity with existing low-level infrastructure, as the
    ↪ language must be suitable for kernel development. It is also
    ↪ pretty much guaranteed that LLVM IR will be the target (and I
    ↪ want to look at Clang's link-time optimizations when it uses LLVM
    ↪  IR). If I am using LLVM IR, the easiest approach will be to
    ↪ support interconnectivity between my language and LLVM IR, rather
    ↪  than offering direct integration with any other programming
    ↪ languages. LLVM IR also supports inline assembly really
    ↪ effectively (and this should be adequate for kernel development
    ↪ although I would have to look into this). My compiler might have
    ↪ to do some analysis of outputted LLVM IR files and even do some
    ↪ minor mutations to them to facilitate my performance
    ↪ optimizations (I think this is likely given that we may link
    ↪ foreign LLVM IR code and we may need to touch this code to get
    ↪ the best out of our optimizations across module boundaries, which
    ↪  is what I would want to consider this an absolute success).

Of course, this final year project is really just initial research and
    ↪ development of something that could take years of work. So I must
    ↪  manage my expectations and remember that I am not creating a
    ↪ fully-functional programming language that is ready for
    ↪ production use, or anything close to it.

## Project Structure

- `compiler` houses the actual compilation process which takes in
    ↪ language code and spits out a final executable.
- `extension` houses the VS Code extension that offers syntax
    ↪ highlighting to the programmer.
- `XXXX_syntax_ideas` contains some early syntax ideas ideas which have
    ↪  been significantly deviated from.

## Git Submodules

When cloning this repo, make sure git "submodules" are enabled and are
    ↪ recursively cloned.
```

appendix_generator/source_code/README.md

```
# Appendix Generator - Source Code

This quick and dirty tool takes all files in the codebase (except some
    ↪ explicitly ignored patterns) and dumps their contents into a
    ↪ latex file for easy inclusion in the interim and final report.
```

```
Proper formatting and escape logic was implemented to get this to work
    ↪ nicely.

Run ‘node app.js‘ from this directory and the result will be outputted
    ↪ to ‘appendix_source_code.tex‘.
```

appendix_generator/source_code/app.js

```
import fs from "fs/promises";
import path from "path";
import { fileURLToPath } from "url";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const root_path = path.resolve(__dirname, "../../");
const output_path = path.join(__dirname, "appendix_source_code.tex");

const ignored_paths = new Set([
    "uoe",
    "pnpm-lock.yaml",
    ".gitignore",
    ".vscodeignore",
]);

const ignored_extensions = new Set([
    ".pdf",
    ".png",
    ".tex",
]);

const escape_latex = (text) => {
    return (text
        .replace(/\\/g, "\\textbackslash{}")
        .replace(/\^/g, "\\textasciicircum{}")
        .replace(/~/g, "\\textasciitilde{}")
        .replace(/[{}&%$#_]/g, "\\$&")
    );
};

const escape_verbatim = (text) => {
    // don't ask me how many hours it took to figure this out

    return (text
        .replace(/@/g, "@\\char\"40@")
        .replace(/\\end\{lstlisting\}/g, "@\\textbackslash{}end\{
```

```
    ↪ lstlisting\}@")
        // .replace(/\\/g, "(*@\\textbackslash{}@*)")
        // .replace(/%/g, "(*@\\%@*)");
    );
};

const parse_gitignore = async (path) => {
    const patterns = [];

    try {
        var content = await fs.readFile(path, "utf8");
    } catch (err) {
        console.log("No .gitignore found at:", path);
        return [];
    }

    const lines = content.split("\n");

    for (let line of lines) {
        line = line.trim();

        if ((false
            || line === ""
            || line.startsWith("#")
        )) {
            continue;
        }

        const is_negated = line.startsWith("!");

        if (is_negated) {
            line = line.substring(1);
        }

        patterns.push({ pattern: line, is_negated });
    }

    return patterns;
};

const pattern_to_regex = (pattern) => {
    const is_root = pattern.startsWith("/");

    if (is_root) {
        pattern = pattern.substring(1);
    }
```

```
const is_directory_only = pattern.endsWith("/");

if (is_directory_only) {
    pattern = pattern.substring(0, pattern.length - 1);
}

let regex_str = "";

for (let i = 0; i < pattern.length; i++) {
    const char = pattern[i];

    if (char === "*") {
        if ((true
            && i + 1 < pattern.length
            && pattern[i + 1] === "*"
        )) {
            regex_str += ".*";
            i++;
            if ((true
                && i + 1 < pattern.length
                && pattern[i + 1] === "/"
            )) {
                i++;
            }
        } else {
            regex_str += "[^/]*";
        }
    } else if (char === "?") {
        regex_str += "[^/]";
    } else if ((false
        || ".*+?^${}()|[]\\".includes(char)
    )) {
        regex_str += "\\" + char;
    } else {
        regex_str += char;
    }
}

if (is_root) {
    regex_str = "^" + regex_str;
} else {
    regex_str = "(^|/)" + regex_str;
}

if (is_directory_only) {
    regex_str += "(/|$)";
} else {
```

```
        regex_str += "(/.*|$)";
    }

    return new RegExp(regex_str);
};

const load_gitignore_rules = async (base_dir) => {
    const rules_by_dir = new Map();

    const analyze_directory = async (dir, rel_path) => {
        rel_path ??= "";

        const gitignore_path = path.join(dir, ".gitignore");
        const patterns = await parse_gitignore(gitignore_path);

        if (patterns.length > 0) {
            const normalized_rel_path = rel_path.replace(/\\/g, "/");
            rules_by_dir.set(normalized_rel_path, patterns);
        }

        const entries = await fs.readdir(dir, { withFileTypes: true });

        for (const entry of entries) {
            if ((true
                && entry.isDirectory()
                && entry.name !== ".git"
            )) {
                const new_rel_path = rel_path ? path.join(rel_path,
↪ entry.name) : entry.name;
                await analyze_directory(path.join(dir, entry.name),
↪ new_rel_path);
            }
        }
    };

    await analyze_directory(base_dir);

    return rules_by_dir;
};

const is_ignored = (file_path, _is_directory, rules_by_dir) => {
    const normalized = file_path.replace(/\\/g, "/");
    const segments = normalized.split("/");

    let is_ignored = false;

    for (let i = 0; i <= segments.length; i++) {
```

```
            const dir_path = segments.slice(0, i).join("/");
            const remaining = segments.slice(i).join("/");

            const rules = rules_by_dir.get(dir_path === "" ? "" : dir_path)
    ↪   ?? [];

            for (const { pattern, is_negated } of rules) {
                const regex = pattern_to_regex(pattern);

                if (regex.test(remaining)) {
                    is_ignored = !is_negated;
                }
            }
        }

        if ((false
            || normalized === ".git"
            || normalized.startsWith(".git/")
        )) {
            is_ignored = true;
        }

        for (const part of segments) {
            if (ignored_paths.has(part)) {
                is_ignored = true;
                break;
            }
        }

        for (const ext of ignored_extensions) {
            if (normalized.endsWith(ext)) {
                is_ignored = true;
                break;
            }
        }

        return is_ignored;
    };

    const collect_files = async (dir, base_path, rules_by_dir) => {
        const results = [];
        const entries = await fs.readdir(dir, { withFileTypes: true });

        for (const entry of entries) {
            const full_path = path.join(dir, entry.name);
            const rel_path = base_path ? path.join(base_path, entry.name) :
    ↪   entry.name;
```

```
        if (is_ignored(rel_path, entry.isDirectory(), rules_by_dir)) {
            continue;
        }

        if (entry.isDirectory()) {
            results.push(...await collect_files(full_path, rel_path,
    ↪ rules_by_dir));
        } else if (entry.isFile()) {
            results.push(rel_path);
        }
    }

    return results;
};

const generate_latex = async () => {
    console.log("Collecting all files from:", root_path);
    console.log("Loading .gitignore rules...");

    const rules_by_dir = await load_gitignore_rules(root_path);
    console.log(`Loaded .gitignore rules from ${rules_by_dir.size}
    ↪ directories.`);

    const files = await collect_files(root_path, "", rules_by_dir);
    files.sort();

    console.log(`Found ${files.length} files to include in this report'
    ↪ s appendix.`);

    let latex = "";
    latex += "% This is an auto-generated appendix using my appendix
    ↪ generator script\n";
    latex += "% Date of generation: " + new Date().toISOString() + "\n\
    ↪ n";
    latex += "\\lstset{\n";
    latex += "  breaklines=true,\n";
    latex += "  breakatwhitespace=false,\n";
    latex += "  postbreak={\\mbox{\\textcolor{red}{$\\hookrightarrow$
    ↪ }\\space}},\n";
    latex += "  columns=fixed,\n";
    latex += "  keepspaces=true,\n";
    latex += "  showstringspaces=false,\n";
    latex += "  tabsize=4,\n";
    latex += "}\n\n";

    let i = 0;
```

```
    for (const file_path of files) {
        console.log(`Processing: ${file_path}`);

        const full_path = path.join(root_path, file_path);

        const content = await fs.readFile(full_path, "utf8");

        const normalized_path = file_path.replace(/\\/g, "/");
        const escaped_path = escape_latex(normalized_path);
        const escaped_content = escape_verbatim(content);

        latex += "\\vspace{0.5cm}\n";
        latex += "\\noindent\\colorbox{lightgray}{\\parbox{\\dimexpr\\
↪ textwidth-2\\fboxsep}{\\small\\textbf{" + escaped_path + "}}}\n\n
↪ ";
        latex += "\\vspace{0.2cm}\n\n";
        latex += "\\begin{lstlisting}[language={}, escapechar=@,
↪ basicstyle=\\ttfamily\\footnotesize, breaklines=true,
↪ breakatwhitespace=false]\n";

        // change to limit how many files are outputted for debugging
↪ purposes.
        if (i <= 999) { // 17
            latex += escaped_content;
        }

        if (!escaped_content.endsWith("\n")) {
            latex += "\n";
        }

        latex += "\\endlstlisting\n\n";

        i++;
    }

    await fs.writeFile(output_path, latex, "utf8");
    console.log(`\nLatex appendix has been succesfully outputted to: ${
↪ output_path}`);
    console.log(`Total number of files: ${files.length}`);
};

generate_latex();
```

appendix_generator/source_code/package.json

```
{
    "type": "module"
}
```

compiler/README.md

```
# Compiler

This directory contains the main entrypoint to the compiler.

The compilation process is separated into two phases:

- ‘frontend‘ (JavaScript): Parsing + initial analysis and
    ↪ transformations.
- ‘backend‘ (C++): Remaining analysis and transformations + codegen
    ↪ using LLVM.

While I intended to write the ‘backend‘ in JavaScript, the primary
    ↪ interface for LLVM is ‘C++‘-based, so I’m writing this phase in ‘
    ↪ C++‘ for simplicity.

## Setup

- Install ‘pnpm ^9.15.1‘.
```

compiler/backend/Dockerfile

```
FROM ubuntu:24.04

RUN apt-get update

RUN apt-get install -y \
    make \
    # LLVM toolchain
    llvm-17 \
    llvm-17-dev \
    clang-17 \
    lld-17 \
    # One can add cross-compilation dependencies here in future
    && rm -rf /var/lib/apt/lists/*

RUN ln -sf /usr/bin/clang-17 /usr/bin/clang && \
    ln -sf /usr/bin/clang++-17 /usr/bin/clang++

ENV LLVM_CONFIG=llvm-config-17
```

```
WORKDIR /app

COPY . .

RUN mkdir -p /app/out /volume/out

RUN make -j$(nproc) all

RUN chmod +x live.sh

VOLUME ["/volume/out"]
VOLUME ["/volume/in"]

# Final steps that are dynamic (using live volumes) are delegated to
   ↪ this dedicated shell script.
CMD ["bash", "./live.sh"]
```

compiler/backend/Makefile

```
CXX                 := clang++
AR                  := ar

CXXFLAGS            := -std=c++20 -O2 -Wall -Wextra -Wpedantic
LDFLAGS             :=

LLVM_COMPONENTS     := core support irreader

LLVM_CXXFLAGS       := $(shell $(LLVM_CONFIG) --cxxflags)
LLVM_LDFLAGS        := $(shell $(LLVM_CONFIG) --ldflags)
LLVM_LIBS           := $(shell $(LLVM_CONFIG) --libs $(LLVM_COMPONENTS))
SYSTEM_LIBS         := $(shell $(LLVM_CONFIG) --system-libs)

BIN                 := bin/backend

src_files := $(wildcard src/*.cpp)
obj_files := $(patsubst src/%.cpp,build/%.o,$(src_files))

build/%.o: src/%.cpp | build
    $(CXX) $(CXXFLAGS) $(LLVM_CXXFLAGS) -c $< -o $@

$(BIN): $(obj_files) | bin
    $(CXX) -o $@ $^ $(LDFLAGS) $(LLVM_LDFLAGS) $(LLVM_LIBS) $(
   ↪ SYSTEM_LIBS)

build:
```

```
    mkdir -p build

bin:
    mkdir -p bin

.PHONY: all
all: $(BIN)
```

compiler/backend/README.md

```
# Compiler Backend

Language: C++.

## Requirements

- Docker
- Linux or WSL (Windows)

## Build & Run

**Linux or WSL**:

```
chmod +x ./host/run.sh
./host/run.sh
```
```

compiler/backend/host/build.sh

```
docker build -t davidcallanan--mu-fyp--compiler-backend .
```

compiler/backend/host/buildrun.sh

```
./host/build.sh && \
./host/run.sh
```

compiler/backend/host/run.sh

```
docker run -it -v "$(pwd)/in:/volume/in" -v "$(pwd)/out:/volume/out"
    ↪ davidcallanan--mu-fyp--compiler-backend
```

compiler/backend/live.sh

```
shopt -s nullglob

/app/bin/backend

files=(/app/out/*)
if (( ${#files[@]} )); then
  cp -r "${files[@]}" /volume/out/
fi
```

compiler/backend/src/main.cpp

```
#include <fstream>
#include <iostream>

int main() {
    std::string output_path = "/app/out/test.txt";

    std::ofstream out_file(output_path);

    if (!out_file) {
        std::cerr << "error" << std::endl;
        return 1;
    }

    out_file << "Hello, world!" << std::endl;

    out_file.close();

    std::cout << "done" << std::endl;

    return 0;
}
```

compiler/ec.js

```
import { Command } from "commander";
import { process as frontend_process } from "./frontend/process.js";

const program = new Command();

const backend_process = () => { };

const compile_module = async (config) => {
    console.log("Compiling module located at:", config.src);
    const result = await frontend_process(config);
```

```
    console.dir(result, { depth: null });
    await backend_process(result);
};

program
    .name("ec")
    .description("Essence C compiler")
    .version("1.0.0");

program
    .command("compile <src> <dest>")
    .description("Compile a module located at <src> and output build
    ↪ artifacts to <dest>")
    .action(async (src, dest, _options) => {
        await compile_module({
            src,
            dest,
        });
    });

program.parse();
```

compiler/frontend/README.md

```
# Compiler Frontend

Language: JavaScript.
```

compiler/frontend/create_fs_source_scope.js

```
import { create_unsuspended_factory } from "./uoe/
    ↪ create_unsuspended_factory.js";

class FsSourceScope {
    _init(dependencies, base_path) {
        this._dependencies = dependencies;
        this._base_path = base_path;
    }

    async resolve(path) {
        // todo: add lru cache of like 1000 entries here.
        const directory = this._dependencies.path.join(this._base_path,
    ↪  path);

        try {
```

```
            var files = await this._dependencies.fs.readdir(directory);
        } catch (e) {
            console.warn(e);
            return undefined;
        }

        if (files.some(file => file.endsWith(".ec"))) {
            return {
                type: "SOURCE MODULE",
                files: files.filter(file => file.endsWith(".ec")),
            };
        } else if (files.some(file => file.endsWith(".eh"))) {
            return {
                type: "INTERFACE MODULE",
                files: files.filter(file => file.endsWith(".eh")),
            };
        }

        return undefined;
    }
}

export const create_fs_source_scope = create_unsuspended_factory(
    ↪ FsSourceScope);
```

compiler/frontend/create_source_scope.js

```
import { create_unsuspended_factory } from "./uoe/
    ↪ create_unsuspended_factory.js";

const validate_redirect = (redirect) => {
    if (false
        || !Array.isArray(redirect)
        || typeof redirect[0] !== "string"
        || (true
            && redirect[1] !== undefined
            && typeof redirect[1] !== "function"
            && typeof redirect[1] !== "object"
        )
        || (true
            && redirect[2] !== undefined
            && typeof redirect[2] !== "string"
        )
    ) {
        return false;
    }
```

```
    return true;
};

const validate_redirects = (redirects) => {
    if (false
        || !Array.isArray(redirects)
        || redirects.some(redirect => !validate_redirect(redirect))
    ) {
        throw new Error("Validation error.");
    }
};

const validate_path = (path) => {
    if (false
        || typeof path !== "string"
        || !/^[a-z0-9_\/.]*$/.test(path)
        || path.includes("..") // prevent user from escaping sandbox.
    ) {
        throw new Error("Validation error.");
    }
};

class SourceScope {
    _init(redirects) {
        validate_redirects(redirects);

        // the goal is to get the longer prefixes first to ensure
    ↪ correct precedence.
        this._redirects = redirects.sort((a, b) => b[0].length - a[0].
    ↪ length);
    }

    resolve(path) {
        validate_path(path);

        for (const [prefix, parent_scope, replacement] of this.
    ↪ _redirects) {
            if (path.startsWith(prefix)) {
                if (false
                    || parent_scope === undefined
                    || replacement === undefined
                ) {
                    return undefined;
                }

                const remainder = path.slice(prefix.length);
```

```
                const new_path = replacement + remainder;

                return parent_scope.resolve(new_path);
            }
        }

        return undefined;
    }
}

export const create_source_scope = create_unsuspended_factory(
    ↪ SourceScope);
```

compiler/frontend/process.js

```
import { mapData, join, opt, multi, opt_multi, or, declare } from "./
    ↪ uoe/ec/blurp.js";

import * as fs from "fs/promises";
import * as node__path from "path";
import { create_fs_source_scope } from "./create_fs_source_scope.js";
import { create_source_scope } from "./create_source_scope.js";

// LEXICAL TOKENS

const WHITESPACE = /^\s+/;
const SINGLELINE_COMMENT = mapData(/^\s*;(.*?)(\n|$)\s*/, data => data.
    ↪ groups[0]);
const SKIPPERS = opt(multi(or(SINGLELINE_COMMENT, WHITESPACE)));
const CORE_SKIPPERS = opt(multi(WHITESPACE));

const withSkippers = (p) => mapData(join(SKIPPERS, p, SKIPPERS), data
    ↪ => data[1]);
const withCarefulSkippers = (p) => mapData(join(SKIPPERS, p,
    ↪ CORE_SKIPPERS), data => data[1]);
const withLeftSkippers = (p) => mapData(join(SKIPPERS, p), data => data
    ↪ [1]);
const withRightSkippers = (p) => mapData(join(p, SKIPPERS), data =>
    ↪ data[0]);

const KW_FORWARDING = withCarefulSkippers("forwarding");
const KW_RESET = withCarefulSkippers("reset");
const KW_TYPE = withCarefulSkippers("type");
const KW_TYPES = withCarefulSkippers("types");
const KW_MAP = withCarefulSkippers("map");
const KW_IMPORT = withCarefulSkippers("import");
```

```
const KW_MOD = withCarefulSkippers("mod");
const LBRACE = withCarefulSkippers("{");
const RBRACE = withCarefulSkippers("}");
const PATH_OUTER_ROOT = withCarefulSkippers(mapData(/^\/\/[a-z0-9_
    ↪ \/\.]*/, data => data.groups.all));
const PATH_MODULE_ROOT = withCarefulSkippers(mapData(/^\/[a-z0-9_
    ↪ \/\.]*/, data => data.groups.all));
const PATH = or(PATH_OUTER_ROOT, PATH_MODULE_ROOT);
const OUTER_ROOT = withCarefulSkippers("//");
const SEMI = withRightSkippers(SINGLELINE_COMMENT);
// i need to go through all these again. why did I put "_" outside the
    ↪ []. why is underscore and numbers not matched in the first part?
const TYPE_IDENT = withCarefulSkippers(mapData(/^(?:([a-z]+(?:_[a-z0
    ↪ -9]+)*::)*[A-Z][a-zA-Z0-9]*)|i[1-9][0-9]{0,4}|u[1-9][0-9]{0,4}|
    ↪ f16|f32|f64|f128/, data => data.groups.all));
const TYPES_IDENT = withCarefulSkippers(mapData(/^(?:([a-z]+(?:_[a-z0
    ↪ -9]+)*::)*(?:[a-z]+(?:_[a-z0-9]+)*))/, data => data.groups.all));
const MOD_IDENT = withCarefulSkippers(mapData(/^(?:([a-z]+(?:_[a-z0
    ↪ -9]+)*::)*(?:[a-z]+(?:_[a-z0-9]+)*))/, data => data.groups.all));
    ↪  // identical to TYPES_IDENT for now.
const SYMBOL = withCarefulSkippers(mapData(/^:([a-zA-Z_][a-zA-Z0-9_]*)
    ↪ /, data => data.groups.all));
const EXTAT = withLeftSkippers("@");
const INTEGER = withCarefulSkippers(mapData(/^[0-9]+/, data => BigInt(
    ↪ data.groups.all)));
const FLOAT = withCarefulSkippers(mapData(/^[0-9]+\.[0-9]+/, data =>
    ↪ data.groups.all));

// PARSER RULES

const symbol_path = mapData(
    multi(
        SYMBOL,
    ),
    (data) => ({
        type: "symbol_path",
        trail: data.map(entry => entry.substring(1)),
    }),
);

const constraint_map = mapData(
    join(
        LBRACE,
        RBRACE,
    ),
    (data) => ({
        type: "constraint_map",
```

```
    }),
);

const constraint_integer = mapData(
    INTEGER,
    (data) => ({
        type: "constraint",
        mode: "constraint_integer",
        value: data,
    }),
);

const constraint_float = mapData(
    FLOAT,
    (data) => ({
        type: "constraint",
        mode: "constraint_float",
        value: data,
    }),
);

const constraint_semiless = or(
    constraint_map,
);

const constraint_semiful = or(
    constraint_float,
    constraint_integer,
);

const constraint_maybesemi = or(
    constraint_semiless,
    mapData(
        join(constraint_semiful, SEMI),
        (data) => data[0],
    ),
);

const forwarding = mapData(
    join(
        KW_FORWARDING,
        LBRACE,
        opt_multi(
            or(
                mapData(
                    join(KW_RESET, or(PATH_OUTER_ROOT, OUTER_ROOT),
↪ SEMI),
```

```
                    (data) => ({ type: "reset", source: data[0] }),
                ),
                mapData(
                    join(PATH_OUTER_ROOT, PATH, SEMI),
                    (data) => ({ type: "redirect", source: data[0],
    ↪ destination: data[1] })
                ),
            )
        ),
        RBRACE,
    ),
    (data) => ({
        type: "forwarding",
        entries: data[2],
    }),
);

const top_forwarding = mapData(
    forwarding,
    (data) => ({
        ...data,
        type: "top_forwarding",
    }),
);

const type_reference = or(
    mapData(
        TYPE_IDENT,
        (data) => ({
            type: "type_reference",
            mode: "type_ident",
            trail: data.split("::"),
        }),
    ),
    mapData(
        KW_MAP,
        (_data) => ({
            type: "type_reference",
            mode: "anon_map",
            leaf_type: undefined,
            call_input_type: undefined,
            call_output_type: undefined,
        }),
    ),
);

const top_type = or(
```

```
    mapData(
        join(
            KW_TYPE,
            TYPE_IDENT,
            type_reference,
            SEMI,
        ),
        (data) => ({
            type: "type",
            trail: data[1],
            definition: data[2],
        }),
    ),
    mapData(
        join(
            KW_TYPE,
            TYPE_IDENT,
            type_reference,
            constraint_maybesemi,
        ),
        (data) => ({
            type: "type",
            trail: data[1],
            definition: data[2],
            constraint: data[3],
        }),
    ),
    mapData(
        join(
            KW_TYPE,
            TYPE_IDENT,
            constraint_maybesemi,
        ),
        (data) => ({
            type: "type",
            trail: data[1],
            constraint: data[2],
        }),
    ),
);

// a value is just a stricter version of a type, but it's still a type
   ↪ from the compiler's perspective. (value constraint perhaps i'll
   ↪ call it constraint instead of value).

const top_types = mapData(
    join(
```

```
        KW_TYPES,
        TYPES_IDENT,
        or(
            mapData(
                join(TYPES_IDENT, SEMI),
                (data) => ({
                    type: "types_reference",
                    mode: "types_ident",
                    trail: data[0],
                }),
            ),
            mapData(
                join(KW_IMPORT, PATH, SEMI),
                (data) => ({
                    type: "types_reference",
                    mode: "anon_import",
                    path: data[1],
                    forwarding: undefined,
                }),
            ),
            mapData(
                join(KW_IMPORT, PATH, forwarding),
                (data) => ({
                    type: "types_reference",
                    mode: "anon_import",
                    path: data[1],
                    forwarding: data[2],
                }),
            ),
        ),
    ),
    (data) => ({
        type: "top_types",
        trail: data[1],
        definition: data[2],
    }),
);

const top_mod = mapData(
    join(
        KW_MOD,
        MOD_IDENT,
        or(
            mapData(
                join(KW_IMPORT, PATH, SEMI),
                (data) => ({
                    type: "mod_reference",
```

```
                    mode: "anon_import",
                    path: data[1],
                    forwarding: undefined,
                }),
            ),
            mapData(
                join(KW_IMPORT, PATH, forwarding),
                (data) => ({
                    type: "mod_reference",
                    mode: "anon_import",
                    path: data[1],
                    forwarding: data[2],
                }),
            ),
        ),
    ),
    (data) => ({
        type: "top_mod",
        trail: data[1],
        definition: data[2],
    }),
);

const case_ = or(
    mapData(
        join(
            symbol_path,
            type_reference,
            SEMI,
        ),
        (data) => ({
            type: "case",
            symbol_path: data[0],
            type_reference: data[1],
        }),
    ),
    mapData(
        join(
            symbol_path,
            type_reference,
            constraint_maybesemi,
        ),
        (data) => ({
            type: "case",
            symbol_path: data[0],
            type_reference: data[1],
            constraint: data[2],
```

```
        }),
    ),
);

const top_extension = mapData(
    join(
        EXTAT,
        TYPE_IDENT,
        case_,
    ),
    (data) => ({
        type: "top_extension",
        target_type: data[1],
        case: data[2],
    })
);

const top_entry = or(
    top_forwarding,
    top_type,
    top_types,
    top_mod,
    top_extension,
    SEMI,
);

// i wonder if values and maps could be combined?
// but for now, value_map will just be optional after type.

const file_root = opt_multi(top_entry);

// SEMANTIC ANALYSIS

// COMPILER

// OUTER INTERFACE

// class ModuleLoader {
//  constructor() {

//  }

//  obtain_module_files(path) {

//  }
```

```
// }

// create_sources_scope([
//  ["//"], // reset
//  ["/", parent_scope, "/"],
//  ["/banana", parent_scope, "/prefix/on/parent/scope/"],
//  ["//octopus", parent_scope, "//prefix/on/parent/scope/"],

// ]);

const make_structured = (entries) => {
    const structured = {
        structure: entries,
        forwarding: [],
    };

    for (const entry of entries) {
        if (entry.type === "top_forwarding") {
            structured.forwarding = [
                ...structured.forwarding,
                ...entry.entries,
            ];
        }
    }

    return structured;
};

const augment = (data) => {
    const module_source_scope = create_source_scope([
        ["//", data.source_scopes.external, "//"],
        ["/", data.source_scopes.internal, "/"],
    ]);

    const forwarding_source_scope = create_source_scope(data.forwarding
↪ .map(entry => {
        if (entry.type === "reset") {
            return [entry.source];
        } else if (entry.type === "redirect") {
            return [entry.source, module_source_scope, entry.
↪ destination];
        }

        console.warn("Entry:", entry);
        throw new Error("Unhandled case.");
    }));
```

```
    return {
        ...data,
        source_scopes: {
            ...data.source_scopes ?? [],
            module: module_source_scope,
            forwarding: forwarding_source_scope,
        },
    };
};

export const process = async (config) => {
    const path = config.src;
    const actual_path = node__path.resolve(path);
    console.log("[FE] Processing module located at path:", actual_path)
    ↪ ;

    const file_system_source_scope = create_fs_source_scope({
        fs,
        path: node__path
    }, actual_path);

    const internal_source_scope = create_source_scope([
        ["//"],
        ["/", file_system_source_scope, "./"],
    ]);

    const result = await internal_source_scope.resolve("/");

    console.log(result);

    const parse_results = [];

    for (const file of result.files) {
        console.log("Parsing file", file);

        const text = (await fs.readFile(node__path.join(actual_path,
    ↪ file), "utf-8")).replaceAll("\r\n", "\n");
        const result = file_root(text);

        if (false
            || result.success === false
            || result.input !== ""
        ) {
            console.error(result);
            throw new Error(`Failed to parse.`);
        }
```

```
        parse_results.push(result.data);
    }

    console.log("Augmenting module data...");

    const combined = parse_results.flat();

    const structured = make_structured(combined);

    structured.module_path = actual_path;

    structured.source_scopes = {
        internal: internal_source_scope,
    };

    const final = augment(structured);

    return final;
};
```

compiler/package.json

```
{
    "type": "module",
    "dependencies": {
        "commander": "^14.0.2"
    }
}
```

compiler/test/a.ec

```
forwarding {
    reset //;
}

forwarding {
    reset //;
    reset //foo/bar/baz;
    //banana/slop /wanana/bop;
}

type foo::Bar f32;
type foo::bar::Baz Banana;
type foo::bar::Bink foo::Bar;
type foo::Coconut map;
```

```
types egg::plant foo::bar;

types banana import /foo/bar;

types orange import /foo/bar forwarding {
    reset //;
    reset //foo/bar/baz;
    //banana/slop /wanana/bop;
}

mod eggplant import //egg/plant;

mod eggplant import //egg/plant forwarding {
    //pear /pear;
}

; todo why is eggplant2 not working?
```

compiler/test/b.ec

```
forwarding {
    //banana/slop /wanana/bop;
    //banana/vegetable /wanana/gravy;
}

@Bar:test:blah27 f32;

@Bar:foo blah::Wah;

type NewType f32 {}
; type NewType map {}

type NewTypeAgain f32 5;
; type NewTypeAgain map 5;

type AnotherNewType f32 5;
type AnotherNewTypeAlright 5;

type Blah 5.5;

@Car:why_is_this_disappearing i32;
@Car:testing f32 5;
; @Car:testing 5;
```

extension/LICENSE.txt

```
no license given
```

extension/README.md

```
# Extension

This directory houses the VS Code extension for my language, which
    ↪ provides syntax highlighting to the programmer.

It is unlikely that any additional extension features would be worked
    ↪ on, as that would be out of the scope of this project.

## Setup

- Ensure `vsce` command is set up on your system.

## Build

- `vsce package`

## Install

- Within VS Code, right click the generated `.vsix` file in the
    ↪ explorer tree.
- Select "Install Extension VSIX".
- You may need to reload your window to see the changes take effect.

## Structure

- The `language-configuration.json` file contains some fundamental
    ↪ syntax rules that VS Code needs to know about.
- The `syntaxes/essencec.tmLanguage.json` file contains the meat up the
    ↪  syntax and also assigns appropriate colors to the syntax.

## Autogenerated Files

The `yo code` command was used to generate the boilerplate for this
    ↪ directory.

Thus, some files were auto-generated by the CLI.

Some other files were scaffolded by the CLI but have since been
    ↪ modified significantly.
```

extension/language-configuration.json

```
{
    "comments": {
        // symbol used for single line comment. Remove this entry if
    ↪ your language does not support line comments
        "lineComment": ";",
    },
    // symbols used as brackets
    "brackets": [
        [
            "{",
            "}"
        ],
        [
            "[",
            "]"
        ],
        [
            "(",
            ")"
        ]
    ],
    // symbols that are auto closed when typing
    "autoClosingPairs": [
        [
            "{",
            "}"
        ],
        [
            "[",
            "]"
        ],
        [
            "(",
            ")"
        ],
        [
            "\"",
            "\""
        ],
        [
            "'",
            "'"
        ]
    ],
    // symbols that can be used to surround a selection
```

```
    "surroundingPairs": [
        [
            "{",
            "}"
        ],
        [
            "[",
            "]"
        ],
        [
            "(",
            ")"
        ],
        [
            "\"",
            "\""
        ],
        [
            "'",
            "'"
        ]
    ]
}
```

extension/package.json

```
{
  "publisher": "blabidy",
  "name": "essence-c",
  "displayName": "essence-c",
  "description": "",
  "version": "0.0.1",
  "engines": {
    "vscode": "^1.96.2"
  },
  "categories": [
    "Programming Languages"
  ],
  "contributes": {
    "languages": [
      {
        "id": "essencec",
        "aliases": [
          "Essence C",
          "essencec"
        ],
```

```
        "extensions": [
          ".ec",
          ".eh"
        ],
        "configuration": "./language-configuration.json"
      }
    ],
    "grammars": [
      {
        "language": "essencec",
        "scopeName": "source.essencec",
        "path": "./syntaxes/essencec.tmLanguage.json"
      }
    ]
  },
  "repository": "https://google.com/"
}
```

extension/syntaxes/essencec.tmLanguage.json

```
{
    "$schema": "https://raw.githubusercontent.com/martinring/tmlanguage
    ↪ /master/tmlanguage.json",
    "name": "Essence C",
    "patterns": [
        {
            "include": "#keywords"
        },
        {
            "include": "#strings"
        },
        {
            "name": "comment.line.content.essencec",
            "match": "^\\s*;(.*)"
        },
        {
            "match": "(;)(.*)",
            "captures": {
                "1": {
                    "name": "punctuation.separator.delimiter.essencec"
                },
                "2": {
                    "name": "comment.line.content.essencec"
                }
            }
        },
```

```
        {
            "name": "entity.name.qualified.custom",
            "begin": "\\b([a-z_][a-z0-9_]*)(::)",
            "beginCaptures": {
                "1": {
                    "name": "entity.name.namespace.component.custom"
                },
                "2": {
                    "name": "punctuation.separator.namespace.custom"
                }
            },
            "end": "(?!::[A-Za-z_][a-zA-Z0-9_]*)",
            "patterns": [
                {
                    "match": "[a-z][a-zA-Z0-9_]*",
                    "name": "entity.name.namespace.component.custom"
                },
                {
                    "match": "[A-Z][a-zA-Z0-9_]*",
                    "name": "entity.name.namespace.component.custom"
                },
                {
                    "match": "::",
                    "name": "punctuation.separator.namespace.custom"
                }
            ]
        },
        {
            "name": "storage.type.object.array.java",
            "match": "[A-Z]([a-z0-9_])*"
        },
        {
            "name": "storage.type.object.array.java",
            "match": "\\b(i[1-9][0-9]{0,4}|u[1-9][0-9]{0,4}|f16|f32|f64
↪ |f128)\\b"
        },
        {
            "name": "entity.name.tag",
            "match": "\\b(this|mod)\\b"
        },
        {
            "name": "entity.name.function",
            "match": ":[a-z0-9_]+"
        },
        {
            "name": "entity.name.function",
            "match": ":\\s"
```

```
},
{
    "begin": "(\\/\\/)(([a-z0-9_\\.])+)",
    "beginCaptures": {
        "1": {
            "name": "keyword.control"
        },
        "2": {
            "name": "entity.name.function"
        }
    },
    "end": "(?![a-z0-9_\\/\\.])",
    "patterns": [
        {
            "name": "entity.name.function",
            "match": "[a-zA-Z0-9_\\.]+"
        },
        {
            "name": "keyword.control",
            "match": "\\/"
        }
    ]
},
{
    "begin": "(\\/)(([a-z0-9_\\.])+)",
    "beginCaptures": {
        "1": {
            "name": "storage.type.object.array.java"
        },
        "2": {
            "name": "entity.name.function"
        }
    },
    "end": "(?![a-z0-9_\\/\\.])",
    "patterns": [
        {
            "name": "entity.name.function",
            "match": "[a-zA-Z0-9_\\.]+"
        },
        {
            "name": "storage.type.object.array.java",
            "match": "\\/"
        }
    ]
},
{
    "name": "keyword.control",
```

```
            "match": "\\/\\/"
        },
        {
            "name": "entity.name.tag",
            "match": "@"
        }
    ],
    "repository": {
        "keywords": {
            "patterns": [
                {
                    "name": "keyword.control.essencec",
                    "match": "\\b(in|if|for|return|import|forwarding|
↪ create)\\b"
                },
                {
                    "name": "keyword.modifier.essencec",
                    "match": "\\b(instantiate|reset|types|type|pub|map|
↪ enum|mut)\\b"
                }
            ]
        },
        "strings": {
            "name": "string.quoted.double.essencec",
            "begin": "\"",
            "end": "\"",
            "patterns": [
                {
                    "name": "constant.character.escape.essencec",
                    "match": "\\\\."
                }
            ]
        }
    },
    "scopeName": "source.essencec"
}
```

B Appendix: LLM Prompts

C Appendix: Screenshots