
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.

2 Description of Work Completed

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 [insert reference].

2.2 Literature Review

Firstly, LLVM was found to be a practical compilation target for low-level programming languages due to (1) its target-independent code generation [1] and (2) its integration capabilities with existing programming languages due to ABI interoperability (including some recent improvements) [2]. Furthermore, its extensive optimization passes shared by languages targeting LLVM [3] should make Essence C more 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 [4]. It was noticed that LLVM has experimental support for inline assembly [5], preventing the layer of indirection of a separate compilation unit (such as in kernel development [6]).

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 [7]. The user can experiment with different models to understand which models are best suited for the task at hand. Both

research and programming tasks have benefited from the use of this tool. 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 repository. As such, the usage of this tool has been temporarily suspended pending further assessment.

3 Future Work

References

- [1] “The LLVM Target-Independent Code Generator — LLVM 22.0.0git documentation”. <https://llvm.org/docs/CodeGenerator.html>
- [2] “GSoC 2025: Introducing an ABI Lowering Library - The LLVM Project Blog”. <https://blog.llvm.org/posts/2025-08-25-abi-library/>
- [3] “(Inline Assembler Expressions) LLVM’s Analysis and Transform Passes — LLVM 22.0.0git documentation”. <https://llvm.org/docs/Passes.html#domtree-dominator-tree-construction>
- [4] “Clang C Language Family Frontend for LLVM”. <https://clang.llvm.org/>
- [5] “LLVM Language Reference Manual — LLVM 22.0.0git documentation” <https://llvm.org/docs/LangRef.html#inline-assembler-expressions>
- [6] “GCC Inline Assembly and Its Usage in the Linux Kernel”. <https://dl.acm.org/doi/fullHtml/10.5555/3024956.3024958>

- [7] “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>.

B Appendix: LLM Prompts

C Appendix: Screenshots