**PROJECT PROPOSAL**

*Compiler*

Table of Contents

[Project Abstract 3](#_Toc113300636)

[Conceptual Design 3](#_Toc113300637)

[Proof of Concept 3](#_Toc113300638)

[Background 3](#_Toc113300639)

[Required Resources 3](#_Toc113300640)

## Project Abstract

What the product does and how it does it from a user point of view – at a high level. (You can include screenshot mockup of the interface)

*This project proposes the design and implementation of a compiler for a subset of C. From a user’s perspective, the product functions as a command-line tool where source code written in the custom language can be translated into an executable form. The tool accepts a text file containing arithmetic expressions and basic control flow statements, then processes it through the standard phases of compilation: tokenizing input, parsing into an abstract syntax tree (AST), checking semantic rules, and then producing executable output (initially via an interpreter, with the option to extend to bytecode).*

## Conceptual Design

Technology

• Language: Go (module: c\_compiler\_demo)

Planned project layout

• `cmd/ccdemo/`

- `main.go` — CLI entrypoint; reads example source and runs the pipeline

• `core/`

- `pipeline.go` — orchestrates phases and formats output (colors/indent)

- `lexer/lexer.go` — lexer that tokenizes the C source using ANTLR

- `parser/parser.go` — parser that builds a parse tree (can use goyacc or ANTLR)

- `ast/ast.go` — AST builder producing an AST (node types)

- `sema/sema.go` — semantic analysis inferring types and annotating nodes

- `ir/ir.go` — Intermediate Representation generation using parsed/typed values

- `codegen/codegen.go` — code generation emitting assembly-like text from IR

- `asm/asm.go` — assembler producing an object-file (bytes, symbols, section size)

- `linker/linker.go` — linker emitting an executable (ELF64 entry, PHDR, sections)

• `examples/`

- `add.c` — program to be compiled

OS: Goal should be cross-platform compatibility but limited to x86 for asm simplicity

## Proof of Concept

This proof of concept is in Ocaml rather than Go or Python, but it’s a strong example of a small compiler:

<https://github.com/nlsandler/nqcc2>

The link to the Go Proof of Concept is here:

https://github.com/nicktagliamonte/c\_compiler\_demo

## Background

Compilers take programs written by people and turn them into something a computer can run. This project will create a small compiler for a custom language, covering all the main steps: breaking code into tokens, building a syntax tree, checking for errors, and running the program. The goal is to show how real compilers work, while keeping the design clear and manageable.

Other projects exist in this space. Crafting Interpreters is a popular open-source guide that shows how to build interpreters and bytecode Vms. *Writing a C Compiler* is a more modern guide, and the source for the above proof of concept. Tools like **ANTLR** provide ready-made parsing support. These will serve as references, but this project will write its own code in Go rather than copying existing implementations.

Industrial compilers like GCC, LLVM/Clang, TensTorrents ML compilers, and javac are much larger and more complex. This project will not try to match their power, but it will use some of the same ideas, like splitting the compiler into clear stages. In this way, the project fits between small educational tools and full professional compilers.

## Required Resources

This project requires no unusual resources. Development will be done in Go using standard tools (parser libraries, Git, GitHub) on a regular laptop or department lab machine. No special hardware, datasets, or reserved compute nodes are needed.