# Studying link-time optimizations in programming language development to facilitate the continuum of static and dynamic modules

**David Callanan**

**21444104**

Final Year Project – 2026

B.Sc. Single Honours in Computational Thinking

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the

B.Sc. Single Honours in Computational Thinking.

**Supervisor: Dr. Phil Maguire**

# Contents

## Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of the B.Sc. Computational Thinking qualification, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: ____________________________ Date: ________________

## Acknowledgements

I would like to thank my parents/guardians/siblings/classmate/friends for their . . .

## Abstract

The logic of computer software is primarily expressed using programming languages, but the balance between versatile and efficient programming languages remains a major challenge. Changes to the execution environment of a program can impact performance, or worse, cause incompatibilities that require major refactoring. Developers often maintain multiple branches of a codebase to keep performance to acceptable levels.

# Chapter 1 – Introduction

TODO

test

## Studying link-time optimizations in programming language development to facilitate the continuum of static and dynamic modules

test

# Chapter 2 – Technical Background

TODO

# Chapter 3 – The Problem

TODO

# Chapter 4 – The Solution

TODO

# Chapter 5 – Evaluation

TODO

# Chapter 6 – Conclusion

TODO

# Chapter 7 – Parsing Library

Prior to this project I had already developed a simple parsing library in JavaScript, which allows for:

(1) Parsing terminal rules using regular expressions.

(2) Forming non-terminal rules by combining other rules in the following ways, sufficient to develop any complex grammar:

   (1) “or” making rule optional

   (2) “join” sequencing rules

   (3) “multi” repeating rules zero or more times

   (4) “opt” making rule optional

(3) Map parsed data to custom structures using the “mapData” function.

In addition it was necessary to implement a trace system to debug issues with parse rules.