# Bachelor Thesis Draft

Denis Leskovar

December 2020

## 1 Introduction

Automation of routine tasks tied with software development has resulted in a tremendous increase in the productivity of software engineers. However, the task of debugging a program remains mostly manual chore. This is due to the difficulty of reliably encountering logic-based runtime errors in the code, a task that, to this day, requires the developer's attention and supervision.

Let program $P$ contain a runtime error $E$ that consistently occurs when $P$ is run with arguments $A$. Since the error $E$ is present at runtime and not compile-time, it can be assumed that syntax wise the code is mostly correct. Therefore, any syntax-based error can be ruled out. This, in turn, leaves us with a set of logical errors $\mathbf{E_L}$. Those include wrongly indexed arrays and calculations that lead to either the incorrect result or an altered control flow of the program. Let $E \in \mathbf{E_L}$. As the generality of errors in $\mathbf{E_L}$ appears too complicated to be solved for all programming languages at once, it is necessary to break the problem down for each programming language. This article is concerned with the logical errors of C and C++. Although C is not a subset of C++, the logical errors made in C can be approached similarly to those in C++. Both languages share mostly comparable constructs. Finding the cause of a logical error in a concrete language requires knowing these constructs, their behavior, and their general handling.

To make finding the cause of an error a systematic approach, one might try removing unnecessary statements in the code, thus minimizing the program. Let $P'$ be a minimal variant of $P$ such that $P'$ results in the same error $E$ as $P$ when run with the same arguments $A$. If done carefully and correctly, $P'$ represents the smallest subset of $P$ regarding code size, while preserving the cause of the error in that subset. Upon manual inspection, the developer is required to make less of an effort to find the cause in $P'$ as opposed to $P$.

The minimization of a program can be achieved in numerous ways. In further sections, the article describes and compares three different approaches. The first is based on naive statement removal and its consequences during runtime. The second removes major chunks of the code while periodically testing the generated program's correctness. The third deploys a sequence of code altering techniques, namely slicing and delta debugging.