An Introduction to Compliers

Anoushka Shrivastava
Carnegie Mellon University

Abstract. Compilers are essential tools in the programming landscape. Indeed, they are often called a man's best friend, if the man is a lonely programmer. However, compilers are also horrible for moral support because they harshly point out a programmer's mistakes. To programmers who truly do consider their compilers to be their closest friend, this can be mentally draining, with side effects such as headaches when code submission deadlines approach, fragmented personal relationships projecting from distrust in one's primary companion, and vomiting. However, we propose a cure: compliers. Compliers are the opposite of compilers: they are extremely supportive of your code on the surface and automatically fix your mistakes under the hood. Our paper details a brief introduction to compliers and strategies to implement the best complier possible.

I. Introduction.

Compliers are tools designed to assist programmers while simultaneously boosting their damaged self-esteem from all of their buggy code. While not all languages rely on compilation, the general functionality of compliers can be extended across programming languages and integrated into existing tools. Specifically, the functionality we focus on is compliers' method of providing errors to, and fixing errors for, the user.

II. Error Message Communication

Compliers target a key point of interaction with the programmer: error messages. Error messages can be subdivided into several categories, and each of these categories causes varying levels of frustration. Good compliers have different strategies to deal with each category type. For example, type errors are a common category of errors, and because they can be somewhat confusing to debug, compliers make sure to add an encouraging emoji to the end of the error message. A good complier will also recognize successful type compilation by playing "My Type" by Saweetie.

No matter the error category, compliers must respond with the technique of a "compliment sandwich" to keep the programmer encouraged. They must also only output one error at a time and patiently wait for the programmer to fix the error before outputting a new one. This prevents the programmer from fainting when they see more errors than code written.

III. Error Detection and Fixes

Compliers also have an important secondary component: they automatically fix parts of your code for you without telling you or showing you where you went wrong. While it may seem like a nightmarish guessing game to guess what a complier does to your code, due to the compiler's training process, it is not. Compliers automatically fix code by using a machine learning model that has been trained on buggy code. If, for example, you commit many spelling errors, your complier will learn your bad coding habits over time and remember them during compile-time. Because programmers make plenty of mistakes, the training dataset for this task is quite large, and compliers can get good at guessing what programmers' true intentions are.

IV. Architecture

Complier architecture mimics compiler architecture. The author has unfortunately not taken compiler design courses and lacks the background to dive into a lot of depth. However, adding a neural network can't really hurt. Neither can applying a decision tree.

V. Conclusions

Compliers, unlike compilers, are a supportive tool to reward good (and bad) code with compliments. This has many positive side-effects, such as cheering up the sleep-deprived face on the other side of the screen that looks less clean than their code. By incorporating artificial intelligence, compliers are also a modern upgrade to traditional compilers. Future research should test the improvements in mood that result from using compliers. Studies might also want to confirm the vomiting thing from the abstract.

VI. References

The author relied on past experiences with, and error messages from, compiling her programs.