DEPARTMENT OF COMPUTER SCIENCE | LUNDS TEKNISKA HÖGSKOLA | PRESENTED 2025-05-16

MASTER THESIS An Evaluation of Approaches to Code Formatting
A Case Study on the Cypher Query Language
STUDENTS Tomas Nyberg, Simon Thuresson
SUPERVISORS Emma Söderberg (LTH), Oskar Damkjær (Neo4j)
EXAMINER Görel Hedin (LTH)

How to write a code formatter

POPULAR SCIENCE SYNOPSIS **Tomas Nyberg, Simon Thuresson**

A code formatter is an essential tool for every developer, which makes it all the more frustrating when a language like Cypher, the language used with the Neo4j database, does not have one. In our thesis, we implement a formatter for Cypher and evaluate the best approach to do so.

A code formatter is a tool that makes code more readable by fixing its white space, as shown in the example. Many developers consider it a must-have tool in a their tool belt. So when such a tool is missing, a demand for implementing one arises. But implementing a formatter is not as straightforward as you might think.

```
// Before
match (n) where n.age  >25 return
    n.name
// After
MATCH (n)
WHERE n.age > 25
RETURN n.name
```

Our thesis evaluates three different approaches to implementing a formatter for Cypher, which is, curiously, missing a formatter, despite being used by the world's most popular graph database. In addition to providing guidelines for how to implement a formatter, we provide a complete formatter for Cypher available open source. Moreover, we complement Cypher's currently ambiguous style guide with additional formatting rules.

The three approaches were pretty printing, optimization-based, and layout-based formatting. Each approach was assessed by output quality, performance, and maintainability. To support our development and enhancement of the style guide, we conducted a survey on Cypher styling preferences, and gathered user feedback.

After implementing all three approaches, we concluded that the pretty printer approach is by far the easiest to implement, and its performance is great. However, the output quality is very poor, as it does not try to keep the line width within a certain limit, which is crucial for readability. Because of this, the competition of being the best approach goes down to layout-based and optimization-based. The layout-based approach wins over the optimization-based approach in all three areas we evaluate.

We justify the win, first, by the fact that the user feedback and user survey showed that the layout-based had the best output quality. Second, it relies on a much simpler algorithm, which makes it faster. Third, we also found the layout-based approach to be much easier to implement and maintain, partly because it has fewer lines of code, roughly 200 versus 450. As a result of layout-based approach win over both optimization-based and pretty printing, we conclude that it is by far the best approach for Cypher, and likely also for other languages.