Bartosz Kosakowski - 400028494
Melissa Gonzales - 400018207
Justin Vu - 001403210
CS 4TB3
March 2nd, 2020

Term Project Proposal

PROJECT DESCRIPTION:

Historically, compilers have been expressed as sequential programs. Intuitively, this means that there could be a large time overhead to compile programs with many lines. This is particularly true of programs with complicated parse trees, as reaching the bottom of a single branch can take an extended period of time. If you compound this issue for programs that have multiple, deep parse trees, a sequential compiler can take an unnecessarily long time to compile the program. Given this, the objective of this project is to re-implement the P0 compiler using Golang, and to explore the areas of the original compiler that can be improved by concurrent and parallel processing.

Firstly, we must ensure that our implementation compiles P0 code exactly as the original compiler; that is, there is no discrepancy between the output of our program and the original one. This can be easily tested by simply providing our program with the same inputs as the P0 compiler, and checking that the results are the same. Our compiler must allow for the language rules to be expressed as intended.

Once we are certain that our compiler functions as intended, we can begin testing it to see if there are any speed improvements compared to the original implementation and how large. This can be done by timing how long it takes the original compiler to compile a given set of programs, and then timing how long it takes our compiler to compile the exact same set of programs. Once all of the results are gathered, we can conclude whether our implementation results in a faster performance than the original code. In order to minimize the number of confounding variables, these tests must be run on the same machine.

DIVISION OF WORK:

A fair division of labour encompasses all team members making meaningful contributions to the project's code base, as well as researching ideas that provide the team with new insight that can aid the development process. Specifically, each team member will:

- Research for possible new insight.
- Develop the codebase.
- Creating and running tests to ensure code quality.
- QAing

- Contribute to the creation and design of the poster board.

WEEKLY SCHEDULE:

Date	*Objectives*
March 2nd to 8th	Reviewing resources for information, begin coding project.
March 9th to 15th	Continue coding, begin working on poster board.
March 16th to 22nd	Continue coding and working on poster board.
March 23rd to 29th	Wrapping up code and poster board, testing code.
30th to April 1st	Final code testing, code and poster review, final submissions.

INSIGHT WE HOPE TO GAIN

Our hypothesis is that rewriting the P0 compiler with Golang and its fast and powerful concurrency features will result in improved performance. It will be interesting in seeing how much of an improvement there is or if the original implementation will still somehow be faster. Golang is also touted to be an easy to use language with a good developer experience and this project will be a good exercise in exploring these claims as well.

Furthermore, we will gain insight on parsing structured data so that our machines can comprehend what the user is trying to convey. As we QA our compiler, we hope to understand procedures that go into testing a compiler, such as determining the suitable type of tests for each piece of the compiler. It will be fascinating to see how we determine what type of testing will be implemented for the P0 compiler written in Golang.

SOURCES OF INFO:

- https://www.researchgate.net/publication/311251505_An_exploration_on_lexical_analysis
    - This article explains the process of creating a lexical analyzer, and could be helpful to us by allowing us to follow along with the process.
- https://pdf.sciencedirectassets.com/280203/1-s2.0-S1877050915X00081/1-s2.0-S1877050915007553/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjED

EaCXVzLWVhc3QtMSJHMEUCIQCNmFRUifILJozTesW%2BpoRNBQMe9ygcXZZ0ZiUxDMPiSQIgfCQo8EOrAfJJJbTI3w%2BDCvQ3mzGkk%2F3VCedqK2f%2BTvcqvQMI%2Bf%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARACGgwwNTkwMDM1NDY4NjUiDOMbL2ba%2BUp3p%2F5iNSqRA%2BsAjYPQHEdY75T53Oh0VZE2CrIoJIK393Pt8tMbq1LfM9F1SdQmWJd%2BaP77T4ZMiI%2FUzgpx%2BqBR449As59HtyOJczpCUhmfWO5%2FDOd%2Bb7Idnvb1fsSee%2F1mSkZBeTS7QpcIP%2B5kSP0I7xLxOozLuW2xVZyMPoyC9GnN9Iz%2F0Mr1EOuwC2UT6VynPo%2FYStaHZOwCBtTITA02idPxbGnHZucUjvSGfcN6e2lafZVS8gtCPZuFxSYCcfJXO9udZCOX8tTaAHQgcek3hGf9REzrOvDRMfqdTTEcWjhTo24ujmKudn7T9VGwj0CM2V1Voru664bxH5PcOG4PajOGLEYGTyRz1xQNrksoiu2kiO6ezi%2F6eTHpexqwLMeJY5rAfqrGJRS5j75zuWDUYF1NW40bUhp7m%2BHU9rFYMMIfwi2dasqS9%2BAJhGFa%2FTUUyDLwJkgwggUF8aXCNp8VXlC528fEUSm8bvbrbILOCRyi6YIgsmflrtmSfcNAZvG3EWpNjWthg0h4lgOfcZyRZtLC0wTbczJoMML%2F6%2FIFOusB1M0FUzVPrQAH%2F%2BEGjaLNeHzzORTGQ%2BQRdaDDIxQ%2BC0OhdUBRXgLLp4Bl6%2FxH5hwR%2BkfAxzSUU6U842X7WeodcpJwZX9krGwVhbG0S%2B45wd7u8YWkOjNjLmklsluciUIwkkotT3MVua%2FTehe6Ti19WfWOTqfp9fTgWyWVH07vNIHuGDmFkW7eUI5Xr%2FbDwcQa3sTtW2644o8p5LuEpBEMtToSAEwY6xhjMpKOlinEI106irFzRdMniCDqI%2F22ejxiF9qonBpUiycsdw74ifWv91RMQa7wPxpQmvRvdMiYC9AWKHUGIsszSbquzw%3D%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200301T012759Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTYX7WDEZ6B%2F20200301%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=6670b034751c70a227b39c68db90ac7e7600d428d3324ce7efa4ca9929a5e85e&hash=df0d1b62b737e2230fa2d9f16d2078df1b3049be4ce2971984039cf8a6ba3b02&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S1877050915007553&tid=spdf-a67d008e-f138-4723-96b1-348db6f01f2a&sid=097fdbfd822f144f8278b0358468f9809ab3gxrqa&type=client

    - This article details how a team of professors implemented a parallel lexical analyzer using OpenMP.
- https://l.messenger.com/l.php?u=http%3A%2F%2Fthegrenze.com%2Fpages%2Fservec.php%3Ffn%3D505.pdf%26name%3DParallel%20Execution%20of%20Tasks%20of%20Lexical%20Analyzer%20usingOpenMP%20on%20Multi-core%20Machine%26id%3D1617%26association%3DMcGraw-Hill%26conference%3DAET%26confyear%3D2018&h=AT2sVzv1OHjnarWcaloiKWVeJAAQq7f3rXj7HG0MFXWWRgYb0Q_jokcfwUkSyGRuUFBj3m0N2QNAVcjns2IovXh8nQ-C7Py92fzy0KLyv8N0Tb6rPA9NIAc2LENrQ54CTmR3Qg
    - Another article detailing an OpenMP implementation of a lexical analyzer.
- https://dl.acm.org/doi/10.4018/IJKBO.2018010105

    - Another resource documenting the design and implementation of a parallel C lexical analyzer.
- The P0 Compiler, Jupyter notebook content (files -> 05 Construction of a Parser -> P0.ipynb)
    - Use this to figure out P0's language rules, how it functions, and investigate areas for optimization/parallelization.
- https://golang.org/doc/
    - Documentation for Golang.
- https://www.freecodecamp.org/news/write-a-compiler-in-go-quick-guide-30d2f33ac6e0/
    - A barebones guide of writing a compiler in Go.
- https://medium.com/rungo/anatomy-of-goroutines-in-go-concurrency-in-go-a4cb9272ff88
    - Tutorial on how to use Go's concurrency features.
- https://drops.dagstuhl.de/opus/volltexte/2018/8676/pdf/dagrep_v007_i012_p050_17502.pdf
    - An article describing the challenges faced when testing compilers, and how to bypass them. Also outlines test case reduction, and software verification.
- System Testing a Compiler: https://anniecherkaev.com/2017/06/07/System-Testing-Compiler.html
    - In order to ensure the developed compiler is correct, we must test our compiler. This article provides insight on how to test a compilers, and the approach to system testing versus unit testing.