Laurence Stolzenberg (260657154)

Maxence Hull (260706895)

Milestone 1

Due February 26th 2017

Compiler Design (COMP520)
Presented to Alexandre Krolik

McGill University, School of Computer Science

Decisions and details of our project:

Scanner/Parser: implemented using SableCC.

Why did we choose SableCC? There are multiple reasons why we found SableCC to be the best choice to implement our compiler in the scope of this course. First of all, both team members preferred working with a higher-level programming language, that is to say Java, versus the other tools mentioned in class that worked with C.

Additionally, one of the members of the team had already worked with SableCC, although an older version with notably different syntax, completing a similar project in the scope of another course at another school. Having some experience with how this tool worked, we felt it was the simplest option we had at the start of our project.

Another interesting point is that SableCC, being in Java, worked with a very easy to use integrated development environment that both team members were familiar with, NetBeans, which allowed us to debug and step into the code when we had some difficult to trace issues.

Other decisions for the Scanner and Parser portions of our project:

While we did have a few issues during the construction of our grammar file, used to generate our scanner with SableCC, we had very few decisions to make as the official specifications of Go with

the few modifications specified for our version (GO Lite) had syntax very similar to what was used by SableCC for the grammar file. One thing that we had no choice was to use a very generic term, identifier, for both variable names and type identifiers, which seemed somewhat overly general to us, but seemed to be in line with what Go uses in its specifications. While this does seem inherently logical, as the Go specifications mention that basic type names (int, float64) can actually be used as identifiers, it did force us to keep our grammar very general and in my opinion slightly ambiguous, but we did manage to avoid any permanent shift/reduce conflicts at this state.

Decisions related to the weeding phase:

Nothing unexpected here, as we left as few things as possible to be done during a further "weeding" phase. To be more specific, we have a "weeder" that passes to ensure only one default case is present in a single switch case, and it was designed in such a way that a "switch" contained in a switch case should still be treated in a valid way (i.e. each may have a default case, independently of the parent/child contents). We also check to ensure that "break" and "continue" statements are only contained inside valid contexts (in "for" / "while" loops) during this weeding phase.

Teamwork and separation of duties.

We tried to be as fair as possible while separating the work for this project. Maxence built the "Backbone" of our grammar, while Laurence designed a number of valid and invalid tests for the compiler. Next, both members separated some of the more complex features to implement. As an

example, Maxence handled the “for” and “while” loops, while Laurence handled the function declarations, function calls, and “Switch” statement. Finally, while Maxence handled the “AST”, Laurence worked on this report.