Milestone 2: Teamwork Report

Design Decisions + Contributions

Alexander Iannantuono, Asa Kohn, and William Chien

March 3, 2020

1 Design and Implementation Decisions

1.1 Symbol table

1.2 Typechecking

Given that we used a lot of weaving in our symbol table, this made the typechecking phase easier to design, although there was still a lot of code to write. For each symbol in our symbol table, we include a pointer that points to the node in the AST where the symbol is first introduced in the program, i.e. where it is declared. By declared, we refer to the "nearest" declaration that can be found, as we support shadowing in GoLite.

In regards to writing the typechecker, we wrote short functions to *separate* the different kinds of declarations, statements or expressions. This was so that the main function that would check if that given part of a program type-checked and resulted in a cleaner and easier function. Another example of this would be for error printing. Most, if not all, of the errors have the same format of the form:

```
Error: (line #) Something went wrong with this (part).
```

We say that it *would be* given that we opted to copy-paste the code every time that it was needed. This was due to the fact that some errors needed to output more information than others. In terms of a function, we could implement that as a linked list, but for something as basic as error reporting, it didn't seem worth it to do that.

In terms of code, we wrote a few helper functions to be able to not have to re-use code all the time, which is quite the standard practice, but it saved us a lot of time. This includes functions that would add to the symbol table, or add or remove a scope.

For scoping, we used a linked list to be able to implement scoping in our compiler. For example, ...

There were a few difficulties encountered while writing the typechecker. Firstly, there are a lot of rules to keep track of. This is why we decided on separating the checking of smaller parts of the program to type check, unless if it type checked trivially, as then one could just refer to the smaller function. Secondly, while implementing some of the typechecking rules, Alex realized that we did not actually implement a boolean literal for the first milestone. We just decided to use `0` and `1` in our C code in order to give them a value. This became an issue as we needed to be able to do something with identifiers, such as: `a || b`. This would not work without some other construct to describe the bool identifier, as `2 || 3` would be considered valid, which is not allowed unless we set up some other notion of what we define a `bool` to be considered in our compiler (as opposed to in GoLite).

2 Amending issues from the first milestone

2.1 String literal in lexer

On line 204. Previously, our lexer would match string literals with newline characters as long as they are within the two double quotes. The issue is fixed where newline characters within two double quotes will not be matched.

2.2 Int literal in lexer

On line 199. Previously hexadecimal numbers are matched using:

$$\texttt{0x[0-9a-fA-F]+}[0-9]+$$

Now we use:

$$\texttt{0x[0-9a-fA-F]+}[1-9][0-9]*|0[0-7]*$$

2.3 Rune \' in lexer

On line 188. `'[\^\\']'` is added.

2.4 Line number recording in parser

`$$->lineno = yylineno` on line 231 for TOK_VAR is moved down two lines. Previously, we were recording line number prior to grouping, i.e. `$$ = $3;`

2.5 Weeding

A typo on line 138 is fixed, where exp meant expstmt.

2.6 Pretty printing

traverse_fields(VARS * f) added in pretty printing. On the other hand, segmentation faults occurred when null pointers are left unchecked, therefore, null checks are added to each pretty printing function.

3 Division of labour and team organization

3.1 Dividing the work

Asa decided to work on the symbol table and modeled it after what he did in his assignment.
Alex decided to work on the typechecker and wrote primarily a boilerplate code for the typechecker file.
William wanted to work on the test programs, as he previously wrote a majority of them in M1.
Alex started to work on writing this report alongside the typechecker. However, due to an unexpected illness, William took over the report, and bravely enough, Asa took over the typechecker.

3.1.1 Asa

Asa has completed the symbol table by himself. He also picked up where Alex has left off in the typechecker.

3.1.2 William

Over the fear of M1 for not being able to test our winnipeg compiler adequately robust and to catch a good number of bugs, Will has written roughly 200 invalid test files (non-distinct) and about 100 valid test files; and additionally, a `testscript.sh` that allows us to run through a good number of test files against our compiler quickly. With the consideration of quality over quantity, William tried to go through the goLite and blank identifier specifications slowly and hopefully in greater detail with additional research, then write out all the test files required to test our compiler. Though, whenever a test idea comes to mind, a file may be added to avoid forgetfulness; and since all test files are written over the span of weeks, some test cases may be non-unique or have been considered previously in a different file. In terms of the report, Will kept the majority of what Alex has left off, and filled in additional information when needed.

3.1.3 Alex

In Alex's implementation of the symbol table, he used a bit of weaving, but mainly decided to have every symbol be looked up and perform a linked list-like search, which in hindsight is very wasteful in resources. For example, given some symbol `a`, his code would look through the current scope and search in outer scopes until it found what it was looking for. This differs in this milestone's implementation as here when we look at a symbol in say, some expression, we have a pointer that points immediately to its declaration in the code. This allows for much faster lookup, at the cost of an extra pointer per identifier. Alex's typechecker was similar to that of this one, given that he has written both his own and this group's with the eventual help of Asa.

Alex spent most of his time working on the typechecking from boilerplate code to the testing (hopefully fixing of bugs). As mentioned above, the report was mainly written by him, except for the parts that involved the work of others. Those parts were written by the members involved.

Another portion of this milestone that he decided to work on was fixing the pretty printing invariance from the first milestone. He had been informed that a decent amount of the pretty printing that was tested during grading did not satisfy the following invariance:

```
pretty(pretty(code)) = pretty(code)
```

After some time writing a script to do some basic invariance testing in a shell file, Alex decided to find why some of the files did not respect the invariance defined above. As it turns out, the majority of the issue was the most basic bug that one finds in writing C code: not checking for a null pointer. While traversing parts of the AST, Alex had forgotten to check for a null pointer, causing the pretty printer to either cause a segmentation fault or to (somehow) fail silently and stop printing. We're not sure if this is what triggers the invariance test to yield a "no". If it's done by computer and not by a human-TA, then this would make sense, as it would not print out everything that it needed to print. Another reason that caused the pretty printing invariance was actually using the wrong function to print out fields of a struct: it would print any set of fields as:

```
f1 t1, f2 t2, ..., fk tk
```

Despite that not working for fields of arbitrary type. This yielded a syntax error as soon as it went through to be pretty printed again, yielding the error. This was easily fixed by just putting them on new lines.

The last problem was due to how for-loops were printed. This yielded another syntax error as it thought that an expression was somehow a function call in the form of `f(expr)` where `f := for`. This was due to not correctly putting semicolons (which was only put if a certain part of a for-loop was not null), and this was an easy fix as well.

3.2 Organization

Organizing this time was a bit more challenging than for the first milestone. We originally preferred to meet in person, and have done so for a number of times. However, given the ongoing/evolving global pandemic and school closure, we decided to work remotely at our homes and communicate via various platforms: Matrix via Riot.im, GitHub issues, email, text messaging, and online meetings.

Alex decided to use small to-do lists in the files that he was working on just to keep track of the work he still had to do. Putting `TODO: (thing)` was also very helpful in case he missed something in the big files that he was working on. For other parts of the project, such as writing this report, Alex added small comments tagging either Will or Asa with `@<name>: message` in case there was something where their input was needed or Alex was requesting that they possibly do that part.