Shao Kun Deng - 260638592

Hai Tong Yang -260453583

Dong Kyu (Jason) Kim – 260531174

Generated code

We have decided to generate C++ code, for several reasons. First, C++ has been around for many year and has been worked on by many people. Compared to more recent programming languages, there has been a lot of work towards the optimization of C++ compilers. Second, we chose C++ over low-level code such as assembly code because we lack the knowledge in those language to produce well designed code in a reasonable amount of time. We chose C++ over C because of GoLite's support for unbound slice types. C++ vector library allows us to implement the slice type in a clean fashion and is probably more efficient than any solution we can come up with. We chose C++ over Java, Python or other higher level languages simply because C++ is lower level and should offer better performance.

Code generation pattern

For most of the program's structure and control flow, such as for loops and if statements, we simply use the constructs provided by C++.

Design decisions

1. There are 2 ways to generate code about alias types. One way is to simply generate the code for the underlying type. The other way uses C++ built-in aliasing feature in **typedef** statements. We have decided to go with the first alternative. We have opted for the second option as it provides a more accurate representation of the source code without any performance drawback.
2. To match Go's design, we have decided to make slice into pointers, which means that assignment will only perform a shallow copy and slices will be passed as reference in function calls. In contrast, arrays assignments will be deep copies and will be passed by value in function calls. Therefore, for GoLite arrays, we will use std::array<T> from C++'s standard template library (STL). For GoLite slices, we will use pointers to std::vector<T> from STL.
3. In if-statement, switch-statement and for-loop, it is allowed to have an initial statement which can declare variables that will only exist in the scope of the control block, but can also be shadow by any variable with the same name declared inside the block. To do so, we have decided to envelop all such structures with another scope that contains the initial statement and the control block itself. This way, the initial statement does not interfere with the outside scope (and can shadow variables from outside scope properly) and let variables inside the control block shadow any variable declared in the initial statement.

Implemented

Declarations

1. Function declarations
2. Variable declarations
3. Type declarations(for all types in GoLite)

Statements

1. Control structures (all if-statements, for-loops, switch-statement)
2. Empty statement
3. break and continue
4. Expression statement
5. return
6. declaration statement
7. assignment
8. op-assignment
9. block
10. print/println

Expressions

All expressions except append and type cast

To be implemented

1. Function forward declaration
2. Global variable forward declaration
3. Adjust main function return type from **void** to **int** and add return statement
4. Append expression still buggy
5. Type cast

Benchmark

The programs serving as benchmark are insertionsort.go and fib.go places inside programs/benchmark/