Group 09:
Nathan Sandum 260716168
Jiawen Wang 260683456
Grégoire Moreau 260874685

COMP 520 WINTER 2019: MILESTONE 2

Symbol Table and Type Checking for GoLite
Professor: Alexander Krolik

Symbol table
The symbol table was implemented in symbol.c and symbol.h largely by Jiawen.

The general structure of a SYMBOL is a TYPE (from tree.h) encompassed by some extra information such as the name of the symbol, parameter and return types for functions, *or* another SYMBOL for nested SYMBOLS such as defined types. We also have a SYMBOLLIST which contains a recursive list of SYMBOLs for things like function parameter lists and the fields of struct types. In addition, after every function declaration, we iterated over each symbol in this SYMBOLLIST and put it in the scope of that function.

For SYMBOLs of defined types, a nested chain of SYMBOLs was used, so that each type had an underlying type it points to. This makes resolveType (as well as symbol printing of that type) simpler, having to recurse through the underlying types until it hits a base time, or a symbol of type array, slice, struct, or type. Slices and arrays also have an entryType.

In addition, a SymbolTable type is Hash table with a hash size of 317, with hash function being the one showed in class. After initializing it in main.c, the rest of the program traverses the AST as usual.

Some other helper functions include:

- getSymbol, which will traverse the cactus stack made by the symbol table and attempt to find a symbol via the identifier it's given, throwing an error if it fails to do so
- printSymbol, which prints a symbol in the same format as the reference compiler, printing the chain of symbols for defined types and the fields of structs.
- declaredLocally, which returns true if the given symbol has been declared in the current symbol table. This is called when a new symbol is introduced into that table to prevent redeclarations within the same scope.

Scoping rules
New scopes are created for the following:

- Around the entire symbol table
- Within the global scope, but after base/constant declarations (e.g. int, bool, true)

Group 09:
Nathan Sandum 260716168
Jiawen Wang 260683456
Grégoire Moreau 260874685

- Blocks
- Short declaration initialization for switches, if, and loop statements
- If statement bodies (including else/else ifs), loop bodies, and switch bodies
- Each switch clause
- After function declarations
  - Though our parser parses functions as containing a 'block' for the function body, we manually scope and unscope after function declarations and recurse directly into the statements of the function body. This gives us a chance to put the symbols made from the parameters of function as new variables in the scope of the function body.

Type checking

Type checking was done in typechecking.c and typechecking.h largely between Grégoire and Nathan. Gregoire implemented the foundation of the type checking code, which traversed the AST, applying relevant functions to each branch of the tree. These functions, meanwhile, being implemented as needed by both, with Nathan focussing particularly on typecheck_EXP.

The type checking begins at typecheck_PROG, which initializes the symbol table, and thereto adds the base types, as well as the constants "true," and "false." The function then proceeds to check the topstmt nodes in typecheck_TOPSTMT. This function checks type declarations, var declarations, and function declarations that occur in the global scope. It then passes into the bodies of all the functions, performing typecheck_STMT on each statement. In any case where an expression comes into play, it checks the type with typecheck_EXP.

Test programs

The 30+ invalid test programs were written primarily by Jiawen. Many of them were made directly from following the Milestone 2 specifications, using that as a basis as to what is and is not allowed. The specific problem of each invalid program is noted in a comment at the top of each file.