COMP 520 Compiler Design Milestone 2

Zhu Xiru, 260498154
Yang Tianzi, 260599365
Chen Hongji, 260610116

Design Decisions and Contributions

Contributions:
- Zhu Xiru: Type Checker, Symbol Table
- Chen Hongji: Weeder. Debugging Milestone 1 Errors
- Yang Tianzi: Unit Tester. Debugging Milestone 1 Errors

Symbol table is tree like data structure.
The super scope includes _, true and false. The next level is global scope
Each Hash Table entry is made of a linked list of symbols. So after hashing the id, we search linearly.
This is mainly to save memory space.
The scoping rules are as follows. We have a super scope, composed of predeclared variables
Then the global scope. For any {}, a new scope is created.
For functions, any parameters required will be inserted in the new scope as predeclared variables which cannot be shadowed in the same function.

Furthermore, with if, for and switch statement the scope is in fact

```
if(int i = 0; i < 5; i++)
```

Memory pointers for type checking are all stored in a hash table garbage collectors.

For type checking, we utilized a one pass type checker. A significant amount of stuff was done in type checking such as casting... so weeding was lighter than expected.

One Weeding Design we could not do was the (a) as being invalid for declarations
For example, (a),b := 2, 3 is invalid in Go but valid in ours.
This is one for a more consistent language and 2, because our compiler kills all parenthesis so it is not checkable

In Terms of Type Check we have made, there are too many of them so this is just a taste of what the type checking rules are
i1.go -> Different Types cannot operate together. For example a rune and float type without cast will result in an error
i2.go -> Type Redeclaration
i3.go -> Adding int with strings
i6.go -> Append a type that is different from the slice
i7.go -> Finding Type Alias
i8.go -> Return Type Must Match Function Return Type
i11.go -> Non identifier at for short declaration
i12.go -> Non Declared Variable
i13.go -> Recursive Type Declaration
i14.go -> Invalid Type Cast
i15.go -> Void return when has a non void return type
i16.go -> No new name declared in short declaration
i17.go -> Struct comparison, must have the exact same fields and types
i18.go -> Accessing underscore field in struct
i19.go -> Using redeclarating id as a parameter name

i20.go -> If condition must be boolean type
i21.go -> Switch condition must match the case condition
i22.go -> Recursive declaration
i23.go -> For condition must be boolean
i24.go -> For cannot declare values in the last for statement part
i25.go -> String substraction
i26.go -> Mod cannot be used with float
i27.go -> Using && without boolean
i28.go -> Using float in shift
i29.go -> Variable Redeclaration, But also stress test mass error potential/memory
i30.go -> Using _ as a type faults

In terms of run time, it seems the type checking part is lighter while the tree building is the heavier portion.
Memory use is not very efficient yet but will be optimized.