Type Checker Description

Symbol Table

Our symbol table uses a similar visitor pattern as the one shown in TUT06. On top of that, we implemented three dataclasses `variable`, `function` and `functions` as the value in the symbol table, which variable and function name is the key. `Variable` has an attribute `type` to represent the variable type. `Function` takes a list of parameter types and a return type. All of the `Function` will be wrapped in `Functions`. The purpose is to allow function declaration of the same name but different parameter types.

Type Checker

Our target language is C, which is mostly a strongly typed language. To follow this typing, our compiler enforces the syntax which type must be specified in variable and function declaration.

We mainly check on three aspects, types are specified when declared, types are the same in assignments and types are compatible in operations.

In the declaration of variables, our compiler only accepts the form of `var: type = val`, which `val` must match the `type`. In function declaration, our compiler accepts `def foo(p1:type,p2:type)->ret_t`, which `p1` and `p2` types will be recorded in `function` class, as well as `ret_t` return type. Also, `ret_t` is pushed to a stack. This stack is popped when our type checker sees a `return` statement, then compares the return type with the `ret_t` at the top of the stack. For multiple declaration of function that has the same name, we lookup the function name, and iterate its value `functions`'s child to find if there is duplicate function declaration (same parameter types).

In assignment, we look up the type of variable on the LHS, then compare it with the type of expression on the RHS, which LHS type must be identical to the RHS type.

For the arithmetic operations, the comparison of LHS and RHS is the same as assignment, however, there may be some type conversion (or dominance) on the RHS, such as `int + float-> float`. Then we compare the dominant type with the type on the LHS.

One thing that our type checker does not support is that our function declaration is not allowed to specify `None` return type. In order to implement this, we are looking to implement `None` return type in parser in later sprints.