Project 3: Typechecker

Christopher Tam
Dhruv Mehra

High Level Implementation

Our compiler does multiple passes over the Abstract Syntax Tree to make sure that the program is type-checked.

In the first pass, the compiler creates 3 main symbol tables:

- Alias_table: which contains all the custom aliases to boolean, float and int types.
- Context_table: And then using this alias_table, the compiler traverses vardecl node in the AST to create context_table which maps all the variable identifiers to their types (for functions it maps their return types)
- Function_table: A table that contains, the params list as an ordered key-value pair for each function identifier.

Once these tables have been built, we do another pass over the funcdecls subtree, checking if all the statements in the function define are type-checked, and we also check to see if all the codepaths in the function return the correct return value.

If we make it this far in the code, then we traverse the stmts subtree to make sure that all the statements in the program are type-checked.

The function check_stmts() traverses the tree, calling necessary methods to check that all subtrees are type-checked. If any node in the subtree causes a mismatch, an error is returned to stderr.

Low Level Implementation

There are 2 types of main Tables that are used in the implementation of typechecker.

```
#[derive(Debug)]
pub struct SymbolTable {
   map: HashMap<String, DynamicType>
}
```

```
#[derive(Debug)]
pub struct FunctionTable {
   map: HashMap<Rc<String>, IndexMap<Rc<String>, DynamicType>>
}
```

Alias_table and context_table are of types SymbolTable and function_table is of type FunctionTable.

Moreover we defined these 2 structs to denote the Types:

```
#[derive(Debug, PartialEq, Clone, Copy)]
enum BaseType {
   Integer,
   Float,
   Boolean,
   Array
}

#[derive(Debug, Clone)]
struct DynamicType {
   cur_type: BaseType,
   sub_type: Option<Rc<Box<DynamicType>>>
}
```

Each type id denoted by DynamicType where cur-type defines the basic type of the identifier. And for nested types like arrays of arrays, there is a reference to a linked list of pointed by sub_type.

This nested DynamicType in conjunction with all the tables provide all the necessary information to check the AST in accordance with the rules of Tiger programming language.

The challenging part was to design the structure of code to make sure that we are able to check the correctness of the program by traversing the AST only once, and trying to make most of the code reusable.

We created several testcases of our own to check weird edge cases, like shadowing and multidimensional array assignment etc. Tracking down bugs in the graph traversal was especially hard.