GT CS 4240 Team 1 Phase 2 Project Submission.
Max Virgil, Scott Vermeyen, Currell Berry
11/12/14

Phase 2 Report

How to Run

For convenience, we have included a copy of the project's jar file that can be used to immediately run our compiler (without having to build the project). From the root directory of the submission, run the following command:

```
java -jar TIGGER.jar -v {inputfile.tig}
```

How to Build

This project depends on Apache Maven for management of the build lifecycle.
Maven can be downloaded 'http://maven.apache.org/download.cgi'.

Prereq: You have **maven** installed and added to your classpath.
Prereq: You are in the `project1/` directory.:
1. To compile the Tiger compiler, run
```
mvn
```
2. To use the Tiger compiler on some "inputfile.tig" file with verbose output options (to display the symbol table and other debug info), run
```
java -jar target/TIGGER.jar -v {inputfile.tig}
```

Run with the `-h` option to view other possible output options.

Design Internals - Overview

For Phase 1 of the project, we wrote a parser grammar that constructed an AST. For Phase 2, we traversed this AST in two passes, dividing the work into two distinct parts: generating the symbol table while performing semantic checks, and generating IR code.

Design Internals - Pass 1: Symbol Table and Semantic Checks

The SymbolTable class (found in "src/main/java/cs4240_team1/SymbolTable.java") is our symbol table representation. We used nested symbol tables to handle nested scopes.

Each symbol table tracks its parent symbol table, contains a list of children, and contains a hash table mapping from symbol name to value. There are three different kinds of entries that a SymbolTable can hold: functions (of type FunctionEntry), types (of type TypeEntry), and variables (of type VarEntry).

The parent pointer which SymbolTable objects hold is used for name-lookup purposes. If a given symbol is not found in the current scope's symbol table, the parent pointer enables us to recursively check the current scope's parents for a matching symbol.

> ***Notes on Reading the Symbol Table Output***
> Nested scopes are represented by indentation level.
> Empty symbol tables (corresponding to scopes without any declarations) exist to let our tree structure of symbol tables match the block structure of the input Tiger program, which makes it easier to manage symbol tables from inside the tree walker grammars.

Semantic checking is primarily handled through auxiliary classes that encode Tiger's semantic rules and log errors accordingly.

The tree walker file responsible for simultaneously generating the symbol table and performing semantic checks is:

```
src/main/antlr3/cs4240_team1/SymTableWalker.g
```

Design Internals - Pass 2: Generate IR code

The IR generation pass only occurs after a program is determined to be semantically valid. Knowing that any input into this tree walker would be valid input helped to simplify the code generation process.

Because the IR code generation process creates temporaries, the tree walker also manages an additional symbol table that only holds temporary variables in addition to the symbol tables created in the first pass.

Internally, IR instructions are stored as a list of objects that each have two main components: an op code and a list of arguments. Using these objects along with the symbol table data structures allows us to differentiate between variables of the same name at runtime. However, we print IR code to the console with special formatting. An example IR snippet looks like:

```
_label0:
add, a#0, b#1, _t0
```

Where `a` and `b` are variables in the user's code, and `_t0` represents a temporary variable. The `#0` and `#1` are unique identifiers to differentiate between two variables of the same name when printing IR code. Note that underscores are used in front of temporaries and labels created by the tree walker to avoid naming conflicts with user-defined symbols (this is also why `#` is used for unique variable identifiers).

The tree walker file responsible for generating the IR code is:

```
src/main/antlr3/cs4240_team1/IRCodeWalker.g
```

Tiger Test Programs

We wrote a variety of Tiger test cases for Phase 2, testing both semantically valid and invalid input. Our test cases are in the `tiger-samples` directory.

We have a number of targeted semantic test cases in:

```
tiger-samples/semantic-tests/invalid/
```

We have a number of targeted IR tests in:

```
tiger-samples/ir-code/simple/
```

Additionally, our large program from phase 1 has been adapted for Phase 2 to be semantically valid and is available at:

```
tiger-samples/large_sample.tig
```