Programming Assignment 3
Due October 30th, 2017 at 11:59pm

1 Overview of the Programming Project

Programming assignments 1-3 will direct you to design and build an interpreter for L. Each assignment will cover one component of the interpreter: lexical analysis, parsing, interpreting L and performing type inference on L.

For this assignment, you will build an interpreter for the L programming language. The specification of L is given as operational semantics in the L reference manual and your interpreter must conform to this specification. You may work either individually or in pairs for this assignment.

2 Abstract Syntax Tree

The data structures for the abstract syntax tree you will build for L ca be found in the `/ast/` sub-directory. You will need to understand the interface of the AST nodes to complete this assignment. The AST nodes are identical to the ones you used in the last assignment. Please refer to the AST source code and the handout of PA2 for any further details.

3 Symbol Table

For this assignment, you are provided with a symbol table implementation. The symbol table functions like the environment E in the operational semantics. In other words, you will add bindings when you encounter a let binding and look up bindings when you encounter identifiers. Since an identifier always refers to the most recently defined identifier with this name, the symbol table support pushing and popping contexts. If you push a context, add bindings and then pop, all bindings added since the push will be removed. You will probably want to push and pop every time you process a let expression. For more details, see the source code of the SymbolTable.

4 Files and Directories

To get started, create a directory where you want to do the assignment on any UT Austin computer science computer and execute the following command in that directory:

```
/projects/cs345.tdillig/PA3/get-assignment
```

This command will copy a number of files to your directory. The only file you will need to modify for this assignment is `Evaluator.cpp` This file contains a skeleton for an interpreter for L. You can actually build an interpreter with the skeleton description, but it only evaluates very few constructs. You should understand all the provided constructs before adding your own and carefully read the `L` manual.

4.1 Building, Running and Testing your Interpreter

To compile your interpreter, simply type

```
make
```

in your directory. This will build a binary called `l-interpreter` that you can run. For example, to run your interpreter on the file `test.L`, you type:

```
./l-interpreter test.L
```

Once your interpreter is finished, this should evaluate L programs. For your reference, you can also run a reference interpreter whose binary we provide. To run the reference interpreter, type:

```
/projects/cs345.tdillig/l-interpreter test.L
```

A big component of this assignment is to test your interpreter thoroughly. It is your responsibility to ensure your interpreter follows the specification in the L manual exactly and never crashes. You will want to feed many different inputs to your interpreter to test it.

4.2 Turning in and Grading

You must hand in the following for this assignment:

- The file `Evaluator.cpp` containing your interpreter
- Five interesting test cases to test your interpreter in a file called `tests.L`

For this assignment, you will receive 20% credit for your test cases and 80% for your interpreter. We will test your interpreter automatically on the best selection of submitted test cases, so it is very much in your interest to have your tests included.

Important: Since we grade your interpreter automatically, do not print anything in your final interpreter since this will confuse our grading script and potentially result in a bad grade for you.

For submission, please submit the file `Evaluator.cpp` as well as the file `tests.txt` with all your test cases. Separate different tests by a newline with three (3) dashes, i.e. `---`.