University of Toronto
CSC467F Compilers and Interpreters, Fall 2018

Assignment 1: Lexical Analyzer

In the Assignment 1, you are required to build a lexical analyzer to scan MiniGLSL, and implement the "trace scanner" functionality (`-Tn` switch) of the compiler, as given in the compiler man page (try "`make man`" in the "compiler467" directory).

1 Starter Files

The starter code tarball is available for download from the course web page. It contains the following source files:

- `compiler467.c` - The main module for the course project.
- `Makefile` - The Makefile for the project.
- `scanner.l` - The dummy `flex` scanner.
- `parser.y` - The dummy `bison` parser.
- `compiler467.man` - The man page for the compiler.
- `globalvars.c` - The global variables.
- `common.h` - The global definitions.

2 Specifications of the Scanner

Your lexical analyzer should store appropriate information for each token identified into a tracing file, and any error recognized into an error file. Submit all source code (not just scanner.l) and add your documentations as comments for each file you modified.

3 Trace Output

When the `-Tn` (trace scanner) switch is activated in the compiler, the global variable `traceScanner` is set to "`TRUE`". Trace information should be sent to the globally visible `FILE *` variable `traceFile`. At this stage, if the input is valid, no other action has to be taken other than that specified above.

Use provided `yTRACE(x)` macro to output the trace. The trace should output the tokens in the same order that they appear in the program.

4 Error Handling

If the scanner encounters an illegal input, it should report an error. Use the yERROR(x) macro to report any errors. Make sure you check for corner cases such as out of bounds integers, and identifiers that exceed the allowed length.

5 Submission

Pack up your code and submit your assignment by typing the following command on one of the UG EECG machines:

```
tar czvf lab1.tar.gz compiler467
submitcsc467f 1 lab1.tar.gz
```

You must not change the directory structure from the provided `starter.tar.gz`.

Submit only one file per two person group from either one of your two accounts. Otherwise, a random partner's submission will be selected.

6 Tips

Below are some helpful hints that will help you with this lab.

- You have to define your tokens in the `parser.y` file. Bison just takes those tokens and defines them in a header file (`parser.tab.h`).
  So if you have `%token integer float` in your `parser.y` file, bison will generate a header file with something that looks like:
  `define integer 112`
  `define float 113`

- It is useful to associate information with some of the tokens. For example, if you have an integer token you would want to also store the value of the integer. This can be done using the `yyval` union. Bison defines `yyval` to be of the `%union` type from the `parser.y` file. You can use `yyval` in your `scanner.l` to store the value of each token.

- You were supplied with a dummy parser that doesn't really do anything. The purpose of the parser is to just repetitively ask the scanner for new tokens. In order for the parser to work properly you MUST add any new tokens you define to the grammar in `parser.y`.