University of Toronto:
CSC467F Compilers and Interpreters, Fall 2014

Assignment 1: Scanner

In phase 1, you are required to build a lexical analyser to scan MiniGLSL, and implement the "trace scaner" functionality (-Tn switch) of the compiler, as given in the compiler man page (try"make man" in the starter1 directory).

Starter Files

The starter code is available for download from the lab web page.
Source files:

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

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), plus a short document describing how you dealt with any special problems encountered (the basic working of flex itself does not have to be described).

Trace output

When the -Tn (trace scanner) switch is activated in the compiler, the global variable traceScaner is set to "TRUE" (1). 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 the provided yTRACE(x) macro to output the trace. The trace should output the tokens in the same order that they appear in the program.

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.

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 purser 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.