CSE 4713/6713 – Programming Languages
Using Flex: The Fast Lexical Analyzer
Assignment #2

Use **flex** to generate the lexical analyzer using the language specification from assignment 1. Upload your source code (i.e., driver.cpp, lexer.h, rules.l). You **need not** upload the lex.yy.c file generated by flex.

Overview of Flex

Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program that recognizes lexical patterns in text. The flex program reads user-specified input files for a description of a scanner to generate. It is traditional to end this file with ".l", for example "rules.l". (The "l" stands for "lex", the name of the original Unix tool. Flex is the open-source replacement. Flex also uses a new algorithm that is much faster than the original lex.) The description is in the form of pairs of regular expressions and C code, called rules. Flex generates a C source file named "lex.yy.c", which defines the function yylex(). The file "lex.yy.c" can be compiled and linked to produce an executable. When the executable is run, it analyzes its input for occurrences of text matching the regular expressions for each rule. Whenever it finds a match, it executes the corresponding C code.

Installing Flex

First, see if it is already installed:

- Open a command (terminal) window and type "which flex" or "flex --version". It might already be there.

Next, install Flex if necessary:

- On Windows, it is best to install Cygwin (http://cygwin.com/). Once the base Cygwin is installed, you can install various Linux tools, such as compilers, editors, and so forth. Flex may be installed when you install the C/C++ compiler (called gcc), or you may have to install it separately.
- On a Mac, you get flex (and many other things) when you install the Xcode development environment.
- I'm not sure what happens on a Linux box, but flex is a standard Unix tool, and should either be installed by default or easy to find and install.
- You can also build Flex. In my opinion, this is the hard way to do it, but you will certainly learn things.
    - Go to http://flex.sourceforge.net and download flex. The last version that I examined is 2.5.39, but any version should work for this lab.
    - After uncompressing the download, you will see a directory named "flex-2.5.39", or similar.
    - Open a terminal window in this directory, and then:
        - First type "./configure" to create a Makefile.
        - Then type "make" to build flex.

- Then type "make install" to install flex. Sometimes you must type "sudo make install" and then type your password. However, note that installing flex in this way will likely overwrite whatever flex came with your system or development environment. This is why it is better to use the flex that comes with Cygwin or Xcode…it should be compatible with the rest of the tools in those environments.
- After executing these steps, flex should be successfully installed. If you have problems, you need to consult the file INSTALL, and start debugging.

Using Flex

Next, examine the sample "rules.l" file:

- "rule.l" contains the rules of how to recognize and return a token when reading each lexeme from the input. Your rules should encode the BullyC language from assignment 1.
- Flex has its own format, go to http://flex.sourceforge.net/manual/ and read sections 3 to 6 to get a basic idea of how to write a rules file.

Using flex to generate a lexical analyzer:

- Type "flex rule.l" in the terminal. Flex will then generate a "lex.yy.c" file in the current path. (If you have issues with the flex command, you may need to install bison as well)
- Then create a project using 'lex.yy.c', 'lexer.h' (containing your token definitions), and 'driver.cpp' (the driver file, given). I have additionally given you a makefile, which will automatically compile this assignment, including calling flex, based on which files you have updated with an editor. But, to use it, you have to compile on the command line instead of a development environment.
- To test your program, use the "sample.in" file. You should get the same results as in assignment 1.

Once it works, name your files `rules.l` and `lexer.h`, and go see the TA.

Once your solution is complete and has been demonstrated to the TA, you only need to submit the files `driver.cpp`, `rules.l`, and `lexer.h`.