CS403: Programming Languages

Lexical Analyzer Module

Introduction

This is your second module for your Designer Programming Language. You may develop your code using any of the allowed procedural languages, but you must ensure it runs correctly under Linux before submission.

Your task is to write a scanner for your programming language. A scanner picks through the source code of a program, identifying the individual tokens. To scan a file, it is necessary to understand lexical analysis. So start by reading *this*.

Specifics

Specifically, you are to write a scanner for your Designer Programming Language. You should implement, at a minimum, the following classes/modules:

- types
- lexeme
- lexer
- scanner

Your implementation must be based upon one-character-at-a-time input (e.g. you may not use Java's string tokenizer if you are writing in Java). That is, your lex method reads a single character and then analyzes what to do with it as above. If you are the least bit confused about this requirement, see me.

One should be able to run the scanner on a file of source code by typing in the command:

```
scanner <f>
```

where `<f>` stands for the name of the file to be scanned.

Additional requirements

You need to support comments in your source code. Comments are considered whitespace.

You are to supply a makefile that responds to the following targets:

`make scanner`	- builds your lexical analyzer
`make test`	- runs your lexical analyzer on a complex source code file

Coding Standards

You must implement your scanner and supporting modules in C, C++, or Java.

Compiling standards:

- Compile at the highest level of warnings (`-Wall` and `-Wextra` for *gcc*, *g++*)
- Supply code that compiles with no warnings or errors
- (C, C++ programmers) *main* returns *int*
- (C, C++ programmers) no implicit typing of functions or variables
- You must supply a makefile

Style:

- (C, C++ programmers) Each "object" should have it's own module/class
- (C, C++ programmers) The public interface for that "object" is placed in the header file
- (C, C++ programmers) Typedef all structures
- (C++ programmers) Use namespaces and "modern include" files

- In general, methods should be not contain very much code - if they do, see if there is some way to reduce the complexity with helper functions - in this particular program, the lex method will be longish because of the switch statement
- There should be no repetitious code
- Use symbolic constants (no numeric contants in code or declarations)
- Programs should not contain explicit limits

Portability:

- Do not supply code that is tied to a specific operating system
- Do not call *system*
- Do not include *conio.h* or any other OS specific header file

Miscellaneous:

- Do not fflush(stdin) - it doesn't do anything useful
- Sign your work (all files)
- Comment appropriately (not too much, not too little, but just right)
- Indent no more than four spaces
- Don't change the length of a tab in your editor
- *Do not read unlimited length strings into fixed length buffers!*

Submitting the assignment

To submit your assignment, delete all object or class files from your working directory, leaving only source code, a makefile, a readme file, and any test cases you may have. Then, while in your working directory, type the command:

```
submit proglan lusth lexer
```

The submit program will bundle up all the files in your current directory and ship them to me. This includes subdirectories as well since all the files in any subdirectories will also be shipped to me. You will receive a weak pass if you supply extraneous files (such as .o files or .class files), so be careful.

Make sure you that I will be able to call your scanner with the command named *scanner*. The command *scanner* must take a filename as a command line argument.

You must supply a makefile that will compile your scanner when responding to the commands *make* and *make run*. The *make* command should build the executable, while the *make run* command should run your lexer on some test cases of your own devising. If you do not supply a makefile, you will receive a failing grade for this assignment.

You may submit as many times as you want before the deadline; new submissions replace old submissions.

You must sign your work and give credit where credit is due!