Compiler Design - WS 2015/16, TU Berlin - Lecturer: Dr. Biagio Cosenza

Exercise 1: Implementing a Lexer

The first assignment is the construction of a lexical analyzer for a shading language known as RTSL (Ray Tracing Shading Language), used in the context of high-quality image synthesis. You should use the lexical analyzer generator Flex. A source file for RTLS, as quite common in computer graphics, is called *shader.*

Inside the given tar archive you will find

- Three `*.rtsl` shader files
- One expected output (`sphere.out`) of the lexical analyzer for the input file (`sphere.rtsl`)
- A paper (`rtsl.pdf`) describing the RTSL language specification. You must read the whole sections 3, all the listed examples and table 1, **before** to start the assignment

As already presented in the lecture on lexical analysis, the lexical analyzer generator (FLEX) is a program that returns tokens and places the lexeme (value of the token) in a variable visible to the outside world. In this assignment, tokens should be printed in the standard output according to the following specifications.

The lexical analyzer should recognize identifiers, variable qualifiers, built-in functions and other keywords defined by the RTSL language

Lexical Classes

RTLS is derived from GLSL, which is a language mainly based on C. Identifier, number and symbol have the same specification of C. There are no char or string type, neither pointer notation.

The lexical analyzer should recognize the following number types:

- Integer constants (numbers with optional +/-), with token INT
- Floating point constants (.04 1.0 … all with optional +/-), with token FLOAT
- Exponential numbers, i.e. integer or decimal number followed by 'E' and an integer number, with token EXP

Note that "`1.`" is perfectly legal in RTSL.

Keywords and reserved symbols

A number of reserved keywords are supported by the language. The lexical analyzer should be able to recognize all keywords present in the input files, and in particular, to distinguish **types**, **variable** and **scope qualifiers**, and **state variables**. If a keyword is not one of those, then a generic **keyword** token must be chosen.

Types include:

- Default types: `int float bool`

- Vector types: `vec2 vec3 vec4 ivec2 ivec3 ivec4 bvec2 bvec3 bvec4`
- Built-in types: `rt_Primitive rt_Camera rt_Material rt_Texture rt_Light`
- From C: `void`

Qualifiers include variable qualifiers (`attribute, uniform, varying, const`) and class scope variable modifier (`const, public, private, scratch`).

Interface methods and **state variables**, they are listed in the paper, in Table 1. Be careful that RTSL's state variables have prefix "`rt_`". Interface methods will be recognized as identifiers, but state variables will have a specific token (STATE).

RTLS has a "swizzle operator" (from GLSL), i.e. an internal component of a type can be accessed using the "dot" operator (see Section 5.5, pag. 103 of GLSL specifications). For example:

```
vec2 pos;
pos.x = 0;
```

should return the following tokens:

```
TYPE vec2
IDENTIFIER pos
SEMICOLON
IDENTIFIER pos
SWIZZLE x
ASSIGN
…
```

RTSL also defines a number of built-in functions (`inverse inside perpendicular dominantAxis trace hit luminance rand pow min max`) and special keywords (`illuminance ambient`). All of them must be recognized as a generic keyword.

Obviously, the lexer should also recognize standard C keyword such as:

```
break case const continue default do double else enum extern for goto
if sizeof static struct switch typedef union unsigned void while
```

Operators are: `+ * - / = == != < <= > >= , : ; ( ) [ ] { } && || ++ --` and they should be recognized with a specific token (see the sample output file `sphere.out`), e.g., PLUS, MUL, MINUS, DIV, ASSIGN, EQUAL, NOT_EQUAL, LT, LE, GT, GE, COMMA, COLON, SEMICOLON, LPARENTHESIS, RPARENTHESIS, LBRACKET, RBRACKET, LBRACE, RBRACE, AND, OR , INC, DEC.

Like C, the language is case sensitive.

RTSL itself is an extension of GLSL (OpenGL Shading Language). If you find keywords (and types) not present in the paper (e.g., `normalize`), you can find a complete list in the GLSL 4.40 specification, Section 3.6, page 27: https://www.opengl.org/registry/doc/GLSLangSpec.4.40.pdf

Be sure that your lexer understands all keywords present in the three *shader* files.

Error Reporting

You must keep track of your line numbers while you are scanning to report errors. For unrecognized symbols an error message should be provided.

For example:
`ERROR(23): Unrecognized symbol "#"`

where 23 is the line number of the error.

Keeping track of line numbers needs to account for the fact that comments can span multiple lines. Line numbers start with line=1

Comments

Comments consist of text enclosed in /* and */, or any text following the // symbol until the end of the line. No token should be produced for comments. The lexer, upon encountering a comment, should produce the first token following the end of the comment.

Output

We are using a semi-automated evaluation approach, so it is imperative that you match the provided output exactly. All output should go to the standard output (not file).

We compare your output with our expected output using the command:

```
> diff –b our_output your_output
```

for all the provided rtls files and, optionally, with additional ones not too much different with the others.

Additional remarks

To complete this exercise, you should be able to write a lexer generator as the ones provided as example in the previous lectures. Your goal is to write a simple program that repeatedly invokes the lexer and prints out the token (i.e token name and lexeme if needed) that the lexer returns (one token per line). You can make the development and testing part easier by using an automated script or makefile that compiles your file(s) and creates the output.

You should test your lexer on the test program and compare with the reference output.

Submission

- Use the ISIS website
- Only submit the `lex` file
- File name has to be: `lastname1_lastname2_lastname3.lex`, e.g.: `maradona_klinsmann.lex`
- First line of the `lex` file has to include first name, surname and student id, for each group participant, e.g.
  `/* Diego Maradona 10, Juergen Klinsmann 18 */`
- Up to **four** people for group
- The submission deadline is 11:59pm of the due date

Links

- FLEX http://flex.sourceforge.net
- Lecture 2 on Lexical Analysis
- [ALSU] 3.5 (the Dragon book)

Be sure that your project works on the machines available at the TEL building.