University of Toronto
CSC467F Compilers and Interpreters, Fall 2018

Assignment 2: Parser

Introduction

In the Assignment 2, you are required to build a parser using the Bison parser generator. The parser should be able to accept the language generated by the MiniGLSL grammar. Your parser must implement the "trace parser" functionality (-Tp command-line flag) of the compiler (see the man page for details; make man in the starter directory). No semantic analysis or AST construction is required in this assignment.

You must start your Assignment 2 code from your implementation of "scanner.l" and "parser.y" file in Assignment 1.

Testing

It is suggested that you test your parser using tests that are expected to both pass and fail. Submitting tests in a sub-directory will be appreciated, but not required.

Language Details

Integer Literals

Valid integer literals fall in closed range [-32767, 32767]. All integers are in base 10 and you can safely ignore the prefix zeros in the literals. Negative integer literals are handled by the unary negate operator '`-`'.

Float Literals

Valid floating point numbers are exactly those represented by the C `float` type. You need to support the following formats of a floating number:

- number before and after the decimal point, e.g. `123.45`
- starting with the decimal point, e.g. `.45`
- scentific notation, e.g. `1.2E5`, `1.3E-4`. Note that the number after the `E` symbol must be an integer, for instance, `1E1.2` is not a valid float literal.

You do not need to support `NaN` literals. Also, negative float literals are also handled by the unary negate operator.

Identifiers

Valid identifiers must be no more than 32 characters in length that starts with a letter or underscore followed by letters, numbers, or underscores. Identifiers are case-sensitive.

MiniGLSL Language Context-free Grammar

$$
\begin{aligned}
\mathit{program} &\rightarrow \mathit{scope} \\
\mathit{scope} &\rightarrow \text{'\{'}\ \mathit{declarations}\ \mathit{statements}\ \text{'\}'} \\
\mathit{declarations} &\rightarrow \mathit{declarations}\ \mathit{declaration} \\
&\rightarrow \epsilon \\
\mathit{statements} &\rightarrow \mathit{statements}\ \mathit{statement} \\
&\rightarrow \epsilon \\
\mathit{declaration} &\rightarrow \mathit{type}\ \mathit{identifier}\ \text{';'} \\
&\rightarrow \mathit{type}\ \mathit{identifier}\ \text{'='}\ \mathit{expression}\ \text{';'} \\
&\rightarrow \text{'const'}\ \mathit{type}\ \mathit{identifier}\ \text{'='}\ \mathit{expression}\ \text{';'} \\
&\rightarrow \epsilon \\
\mathit{statement} &\rightarrow \mathit{variable}\ \text{'='}\ \mathit{expression}\ \text{';'} \\
&\rightarrow \text{'if'}\ \text{'('}\ \mathit{expression}\ \text{')'}\ \mathit{statement}\ \mathit{else\_statement} \\
&\rightarrow \text{'while'}\ \text{'('}\ \mathit{expression}\ \text{')'}\ \mathit{statement} \\
&\rightarrow \mathit{scope} \\
&\rightarrow \text{';'} \\
\mathit{else\_statement} &\rightarrow \text{'else'}\ \mathit{statement} \\
&\rightarrow \epsilon \\
\mathit{type} &\rightarrow \text{'int'}|\text{'ivec2'}|\text{'ivec3'}|\text{'ivec4'} \\
&\rightarrow \text{'bool'}|\text{'bvec2'}|\text{'bvec3'}|\text{'bvec4'} \\
&\rightarrow \text{'float'}|\text{'vec2'}|\text{'vec3'}|\text{'vec4'} \\
\mathit{expression} &\rightarrow \mathit{constructor} \\
&\rightarrow \mathit{function} \\
&\rightarrow \mathit{integer\_literal} \\
&\rightarrow \mathit{float\_literal} \\
&\rightarrow \text{'true'}|\text{'false'} \\
&\rightarrow \mathit{variable} \\
&\rightarrow \mathit{unary\_op}\ \mathit{expression} \\
&\rightarrow \mathit{expression}\ \mathit{binary\_op}\ \mathit{expression} \\
&\rightarrow \text{'('}\ \mathit{expression}\ \text{')'} \\
\mathit{variable} &\rightarrow \mathit{identifier} \\
\mathit{variable} &\rightarrow \mathit{identifier}\text{'['}\mathit{integer\_literal}\text{']'} \\
\mathit{unary\_op} &\rightarrow \text{'!'}|\text{'-'} \\
\mathit{binary\_op} &\rightarrow \text{'\&\&'}|\text{'||'}|\text{' ==' }|\text{'! ='}\ |\text{' <' }|\text{' <='} \\
&\rightarrow \text{' >' }|\text{' <' }|\text{' +' }|\text{' -' }|\text{' *' }|\text{'/'}|\text{'} \\
\mathit{constructor} &\rightarrow \mathit{type}\ \text{'('}\ \mathit{arguments}\ \text{')'} \\
\mathit{function} &\rightarrow \mathit{function\_name}\ \text{'('}\ \mathit{arguments\_opt}\ \text{')'} \\
\mathit{function\_name} &\rightarrow \text{'dp3'}|\text{'lit'}|\text{'rsq'} \\
\mathit{arguments\_opt} &\rightarrow \mathit{arguments}\ |\ \epsilon \\
\mathit{arguments} &\rightarrow \mathit{arguments}\ \text{','}\ \mathit{expression}\ |\ \mathit{expression}
\end{aligned}
$$

Operator Precedence and Associativity

Table 1 shows the precedence and assiciativity of MiniGLSL operators, from highest to lowest. Left-to-right associativity implies that `1+2+3` is interpreted as `(1 + 2) + 3`.

Right-to-left associativity implies that `1^2^3` is interpreted as `1^(2^3)`. High precedence implies that `-1-2` is interpreted as `(-1)-2`.

Precedence	Operator	Associativity
0	[ ] vector subscript ( ) function call ( ) constructor call	Left-associative
1	! logical negate - arithmetic negate	Unary
2	^ exponentiation	Right-associative
3	* multiply / divide	Left-associative
4	+ add - substract	Left-associative
5	== equal-to != not-equal-to < less-than <= less-or-equal-to > larger-than > larger-or-equal-to	Non-associative
6	&& Boolean AND	Left-associative
7	|| Boolean OR	Left-associative

Table 1: MiniGLSL operator associativity

Submission

First, don't forget to fill in your group information in the `compiler467.c` source code file. Second, pack up your code and submit your assignment by typing the following command on one of the UG EECG machines:

```
tar czvf lab2.tar.gz compiler467
submitcsc467f 2 lab2.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.