Module 1

As part of the first phase of your project you have to implement the syntax analyser for a subset of the decaf programming language. Your analyser should be able to handle simple expressions.
Here is the grammar subset you have to implement

⟨program⟩ → class Program '{' ⟨field_decl⟩∗ ⟨statement_decl⟩∗ '}'

⟨field_decl⟩ → ⟨type⟩ {⟨id⟩ | ⟨id⟩ '[' ⟨int_literal⟩ ']' };

⟨statement_decl⟩ → ⟨location⟩ ⟨assign_op⟩ ⟨expr⟩ ;
| callout (⟨string_literal⟩ [, ⟨callout_arg⟩+,]) ;

where all other (required) fields follow the rules as defined in the decaf manual.

Your analyser should successfully parse all valid programs, and give an error if it encounters an invalid program.
Output (to stdout) “Success” on a successful parse, and “Syntax Error” in case of an error.
~~Output (to stderr) the name of the field reached followed by an endl.~~
~~For example, when processing ⟨statement_decl⟩ → ⟨location⟩ ⟨assign_op⟩ ⟨expr⟩ ;~~
~~write “statement_decl\n” to stderr.~~

Please stick to the output format as there will be no manual evaluation of codes submitted.

More information on submission format and a sample input output file will be coming soon.

Cheers!

Update (12/09/2015):

Create a new file “flex_output.txt” which on encountering a symbol outputs the respective Output value on a new line.

flex_output.txt

Symbol Found	**Output**
boolean	BOOLEAN_DECLARATION
callout	CALLOUT
class	CLASS

false	BOOLEAN: false
int	INT_DECLARATION
true	BOOLEAN: true
<id>	ID: <id>
<int_literal>	INT: <int_literal>
<char_literal>	CHARACTER: <char_literal>
<string_literal>	STRING: <string_literal>

Priority Order (Highest to Lowest, the ones on same level have same priority):

1) Unary Minus
2) !
3) * , / , %
4) + -
5) > <

All operators are “left”-associative.

Similarly, create a new file “bison_output.txt” which on encountering a rule, outputs the respective Output value on a new line

bison_output.txt

<program>	PROGRAM ENCOUNTERED
int <id>	INT DECLARATION ENCOUNTERED. ID=<id>
int <id> [<size>]	INT DECLARATION ENCOUNTERED. ID=<id> SIZE=<size>
bool <id>	BOOL DECLARATION ENCOUNTERED. ID=<id>
bool <id> [<size>]	BOOL DECLARATION ENCOUNTERED. ID=<id> SIZE=<size>
⟨location⟩ ⟨assign_op⟩ ⟨expr⟩	ASSIGNMENT OPERATION ENCOUNTERED
callout (⟨string_literal⟩ [, ⟨callout_arg⟩+,])	CALLOUT TO <string_literal> ENCOUNTERED
<location>	LOCATION ENCOUNTERED=<id>
<int_literal>	INT ENCOUNTERD=<int_literal>
<char_literal>	CHAR ENCOUNTERED=<char_literal>
<bool_literal>	BOOLEAN ENCOUNTERD=<bool_literal>
expr ‘+’ expr	ADDITION ENCOUNTERED
expr ‘-’ expr	SUBTRACTION ENCOUNTERED
expr ‘*’ expr	MULTIPLICATION ENCOUNTERED
expr ‘/’ expr	DIVISION ENCOUNTERED
expr ‘%’ expr	MOD ENCOUNTERED
expr ‘<’ expr	LESS THAN ENCOUNTERED
expr ‘>’ expr	GREATER THAN ENCOUNTERED