

Decaf Language

Formal Decaf Grammar

<program>	→	class Program ‘{’ <field_decl>* <method_decl>* ‘}
<field_decl>	→	<type> { <id> <id> ‘[’ <int_literal> ‘]’ }+, ;
<method_decl>	→	{ <type> void } <id> ([{ <type> <id> }+,]) <block>
<block>	→	‘{’ <var_decl>* <statement>* ‘}
<var_decl>	→	<type> <id>+, ;
<type>	→	int boolean
<statement>	→	<location> <assign_op> <expr> ; <method_call> ; if (<expr>) <block> [else <block>] for <id> = <expr> , <expr> <block> return [<expr>] ; break ; continue ; <block>
<assign_op>	→	= += -=
<method_call>	→	<method_name> ([<expr>+,]) callout (<string_literal> [, <callout_arg>+,])
<method_name>	→	<id>
<location>	→	<id> <id> ‘[’ <expr> ‘]’
<expr>	→	<location> <method_call> <literal> <expr> <bin_op> <expr> - <expr> ! <expr> (<expr>)
<callout_arg>	→	<expr> <string_literal>
<bin_op>	→	<arith_op> <rel_op> <eq_op> <cond_op>
<arith_op>	→	+ - * / %
<rel_op>	→	< > <= >=
<eq_op>	→	== !=
<cond_op>	→	&&
<literal>	→	<int_literal> <char_literal> <bool_literal>
<id>	→	<alpha> <alpha_num>*
<alpha_num>	→	<alpha> <digit>
<alpha>	→	a b c ... z A B C ... Z _
<digit>	→	0 1 2 ... 9
<hex_digit>	→	<digit> a b c ... f A B C ... F
<int_literal>	→	<decimal_literal> <hex_literal>
<decimal_literal>	→	<digit> <digit>*
<hex_literal>	→	0x <hex_digit> <hex_digit>*
<bool_literal>	→	true false
<char_literal>	→	‘ <char> ’
<string_literal>	→	“ <char>+ ”

Formal Decaf Grammar Notation Guide

<foo>	Non-terminal symbol
foo	Terminal symbol
[x]	Zero or one x
x*	Zero or more x
x+,	A comma separated list of one or more x
{ }	Grouping
{‘	note: braces in quotes ‘{’ are terminals
	OR

Considerations

- Whitespace (newlines, tabs, spaces, form feed) may appear between lexical tokens.
- Decaf has single line comments which begin with **//** and end in a **newline**.
- Both whitespace and comments should be skipped by the lexer.
- Reserved words (if, for, else, ...) and identifiers are case-sensitive.
- Numbers in Decaf are 64-bit signed integers (hexadecimal or decimal).
- <char> is any ASCII character between 32 and 126 (decimal).
- <char> can also be any of the 2-character sequences: **\"**, **\'**, ****, **\t**, or **\n**.
- Arrays are 1-dimensional and have compile-time fixed size.
- Fields are allocated on the HEAP
- Variables are allocated on the STACK

Order of Operator Precedence (High -> Low)

Operators	Description
-	Unary Minus
!	Logical Not
*, /, %	Multiply, Divide, Modulo
+, -	Add, Subtract
<, <=, >=, >	Relational
==, !=	Equality
&&	Conditional And
	Conditional Or

Semantic Rules

- (1) No identifier is declared twice in the same scope.
- (2) No identifier is used before it is declared.
- (3) The program contains a definition for a method called main that has no parameters (note that since execution starts at method main, any methods defined after main will never be executed).
- (4) The int_literal in an array declaration must be greater than 0.
- (5) The number and types of arguments in a method call must be the same as the number and types of the formals, i.e., the signatures must be identical.
- (6) If a method call is used as an expression, the method must return a result.
- (7) A return statement must not have a return value unless it appears in the body of a method that is declared to return a value.
- (8) The expression in a return statement must have the same type as the declared result type of the enclosing method definition.
- (9) An id used as a location must name a declared local/global variable or formal parameter.
- (10) For all locations of the form id[expr]
 - (i) id must be an array variable
 - (ii) the type of expr must be int
- (11) The expr in an if statement must have type boolean.
- (12) The operands of arith_op and rel_op must have type int.
- (13) The operands of eq_op must have the same type, either int or boolean.
- (14) The operands of cond_op and logical not ! must have type boolean.
- (15) The location and the expr in an assignment, location = expr, must have the same type.
- (16) The location and the expr in an incrementing/decrementing assignment, location += expr and location -= expr, must be of type int.
- (17) The initial expr and the ending expr of for must have type int.
- (18) Break and continue statements must be within the body of a for.