**Fully LL(1) Grammar for the Programming Language**

**Program Structure**

<program> -> <dec\_list> <main> <dec\_list>

<main> -> main ( ) { <mst> }

<dec\_list> -> <dec> <dec\_list> | ε

<dec> -> <struct\_dec> | <var\_dec>

**Structure Declarations**

<struct\_dec> -> <access\_modifier> struct ID <extends> { <struct\_body> }

<access\_modifier> -> public | private

<extends> -> extends ID | ε

<struct\_body> -> <struct\_member\_list>

<struct\_member\_list> -> <struct\_member> <struct\_member\_list> | ε

<struct\_member> -> DT ID <dt\_member\_tail> | constructor ( <param\_list> ) <body>

<dt\_member\_tail> -> ( <param\_list> ) <body> | <var\_init> <var\_dec\_tail> ;

**Member Declarations**

<method\_dec> -> DT ID ( <param\_list> ) <body>

<field\_dec> -> <var\_dec>

<constructor\_dec> -> constructor ( <param\_list> ) <body>

<var\_dec> -> DT <var\_dec\_list> ;

<var\_dec\_list> -> ID <var\_init> <var\_dec\_tail>

<var\_dec\_tail> -> , ID <var\_init> <var\_dec\_tail> | ε

<var\_init> -> = <expr> | ε

**Parameters and Body**

<param\_list> -> <param> <param\_list\_tail> | ε

<param\_list\_tail> -> , <param> <param\_list\_tail> | ε

<param> -> DT ID

<body> -> { <mst> }

**Array Declarations**

<array\_dec> -> DT ID [ ] = { <arr\_items> } ;

<array\_struct\_dec> -> ID ID [ ] = { <arr\_items> } ;

<arr\_items> -> <arr\_item\_list> | ε

<arr\_item\_list> -> <expr> <arr\_item\_tail>

<arr\_item\_tail> -> , <expr> <arr\_item\_tail> | ε

**Statement Structure**

<mst> -> <sst> <mst> | ε

<sst> -> <while\_stmt> | <for\_stmt> | <if\_stmt> | <do\_while\_stmt> |

<try\_stmt> | <throw\_stmt> | <return\_stmt> | <continue\_stmt> |

<break\_stmt> | <var\_dec> | <expr\_stmt>

<expr\_stmt> -> <expr> ;

**Control Structures**

<while\_stmt> -> while ( <expr> ) <stmt\_body>

<stmt\_body> -> ; | <sst> | { <mst> }

<for\_stmt> -> for ( <for\_init> ; <for\_cond> ; <for\_update> ) <stmt\_body>

<for\_init> -> <var\_dec> | <expr> ; | ;

<for\_cond> -> <expr> | ε

<for\_update> -> <expr\_list> | ε

<if\_stmt> -> if ( <expr> ) <stmt\_body> <else\_part>

<else\_part> -> else <stmt\_body> | ε

<do\_while\_stmt> -> do <stmt\_body> while ( <expr> ) ;

**Function Calls and This**

<this\_stmt> -> this <this\_target> ;

<this\_target> -> ID | <func\_call>

<func\_call> -> ID ( <arg\_list> )

<arg\_list> -> <arg> <arg\_list\_tail> | ε

<arg\_list\_tail> -> , <arg> <arg\_list\_tail> | ε

<arg> -> <expr>

**Expressions**

<expr\_list> -> <expr> <expr\_list\_tail>

<expr\_list\_tail> -> , <expr> <expr\_list\_tail> | ε

<expr> -> <assign\_expr>

<assign\_expr> -> <or\_expr> <assign\_expr\_tail>

<assign\_expr\_tail> -> <assign\_op> <assign\_expr> | ε

<assign\_op> -> = | += | -= | \*= | /= | %=

<or\_expr> -> <and\_expr> <or\_expr\_tail>

<or\_expr\_tail> -> OR <and\_expr> <or\_expr\_tail> | ε

<and\_expr> -> <equality\_expr> <and\_expr\_tail>

<and\_expr\_tail> -> AND <equality\_expr> <and\_expr\_tail> | ε

<equality\_expr> -> <relational\_expr> <equality\_expr\_tail>

<equality\_expr\_tail> -> <eq\_op> <relational\_expr> <equality\_expr\_tail> | ε

<eq\_op> -> == | !=

<relational\_expr> -> <additive\_expr> <relational\_expr\_tail>

<relational\_expr\_tail> -> <rel\_op> <additive\_expr> <relational\_expr\_tail> | ε

<rel\_op> -> < | > | <= | >=

<additive\_expr> -> <multiplicative\_expr> <additive\_expr\_tail>

<additive\_expr\_tail> -> <add\_op> <multiplicative\_expr> <additive\_expr\_tail> | ε

<add\_op> -> + | -

<multiplicative\_expr> -> <unary\_expr> <multiplicative\_expr\_tail>

<multiplicative\_expr\_tail> -> <mult\_op> <unary\_expr> <multiplicative\_expr\_tail> | ε

<mult\_op> -> \* | / | %

<unary\_expr> -> <prefix\_op> <unary\_expr> | <postfix\_expr>

<prefix\_op> -> ++ | -- | !

<postfix\_expr> -> <primary> <postfix\_expr\_tail>

<postfix\_expr\_tail> -> <postfix\_op> | ε

<postfix\_op> -> ++ | --

<primary> -> <literal> | <var\_access> | ( <expr> )

<var\_access> -> ID <var\_access\_tail>

<var\_access\_tail> -> <array\_access> | <func\_call\_expr> | ε

<array\_access> -> [ <expr> ]

<func\_call\_expr> -> ( <arg\_list> )

**Exception Handling**

<try\_stmt> -> try { <mst> } <catch\_list>

<catch\_list> -> <catch\_item> <catch\_list\_tail>

<catch\_item> -> catch ( ID ID ) { <mst> }

<catch\_list\_tail> -> <catch\_item> <catch\_list\_tail> | ε

<throw\_stmt> -> throw <throw\_value> ;

<throw\_value> -> <expr> | new ID ( <arg\_list> )

**Other Statements**

<return\_stmt> -> return <return\_value> ;

<return\_value> -> <expr> | ε

<continue\_stmt> -> continue ;

<break\_stmt> -> break ;

**Constants and Literals**

<literal> -> <int\_lit> | <float\_lit> | <char\_lit> | <string\_lit> | <bool\_lit> | null

<int\_lit> -> INT\_CONST

<float\_lit> -> FLOAT\_CONST

<char\_lit> -> CHAR\_CONST

<string\_lit> -> STRING\_CONST

<bool\_lit> -> true | false