C

<translation-unit> ::= {<external-declaration>}\*

<external-declaration> ::= <function-definition>

| <declaration>

<function-definition> ::= {<declaration-specifier>}\* <declarator> {<declaration>}\* <compound-statement>

<declaration-specifier> ::= <storage-class-specifier>

| <type-specifier>

| <type-qualifier>

<storage-class-specifier> ::= auto

| register

| static

| extern

| typedef

<type-specifier> ::= void

| char

| short

| int

| long

| float

| double

| signed

| unsigned

| <struct-or-union-specifier>

| <enum-specifier>

| <typedef-name>

<struct-or-union-specifier> ::= <struct-or-union> <identifier> { {<struct-declaration>}+ }

| <struct-or-union> { {<struct-declaration>}+ }

| <struct-or-union> <identifier>

<struct-or-union> ::= struct

| union

<struct-declaration> ::= {<specifier-qualifier>}\* <struct-declarator-list>

<specifier-qualifier> ::= <type-specifier>

| <type-qualifier>

<struct-declarator-list> ::= <struct-declarator>

| <struct-declarator-list> , <struct-declarator>

<struct-declarator> ::= <declarator>

| <declarator> : <constant-expression>

| : <constant-expression>

<declarator> ::= {<pointer>}? <direct-declarator>

<pointer> ::= \* {<type-qualifier>}\* {<pointer>}?

<type-qualifier> ::= const

| volatile

<direct-declarator> ::= <identifier>

| ( <declarator> )

| <direct-declarator> [ {<constant-expression>}? ]

| <direct-declarator> ( <parameter-type-list> )

| <direct-declarator> ( {<identifier>}\* )

<constant-expression> ::= <conditional-expression>

<conditional-expression> ::= <logical-or-expression>

| <logical-or-expression> ? <expression> : <conditional-expression>

<logical-or-expression> ::= <logical-and-expression>

| <logical-or-expression> || <logical-and-expression>

<logical-and-expression> ::= <inclusive-or-expression>

| <logical-and-expression> && <inclusive-or-expression>

<inclusive-or-expression> ::= <exclusive-or-expression>

| <inclusive-or-expression> | <exclusive-or-expression>

<exclusive-or-expression> ::= <and-expression>

| <exclusive-or-expression> ^ <and-expression>

<and-expression> ::= <equality-expression>

| <and-expression> & <equality-expression>

<equality-expression> ::= <relational-expression>

| <equality-expression> == <relational-expression>

| <equality-expression> != <relational-expression>

<relational-expression> ::= <shift-expression>

| <relational-expression> < <shift-expression>

| <relational-expression> > <shift-expression>

| <relational-expression> <= <shift-expression>

| <relational-expression> >= <shift-expression>

<shift-expression> ::= <additive-expression>

| <shift-expression> << <additive-expression>

| <shift-expression> >> <additive-expression>

<additive-expression> ::= <multiplicative-expression>

| <additive-expression> + <multiplicative-expression>

| <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <cast-expression>

| <multiplicative-expression> \* <cast-expression>

| <multiplicative-expression> / <cast-expression>

| <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>

| ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>

| ++ <unary-expression>

| -- <unary-expression>

| <unary-operator> <cast-expression>

| sizeof <unary-expression>

| sizeof <type-name>

<postfix-expression> ::= <primary-expression>

| <postfix-expression> [ <expression> ]

| <postfix-expression> ( {<assignment-expression>}\* )

| <postfix-expression> . <identifier>

| <postfix-expression> -> <identifier>

| <postfix-expression> ++

| <postfix-expression> --

<primary-expression> ::= <identifier>

| <constant>

| <string>

| ( <expression> )

<constant> ::= <integer-constant>

| <character-constant>

| <floating-constant>

| <enumeration-constant>

<expression> ::= <assignment-expression>

| <expression> , <assignment-expression>

<assignment-expression> ::= <conditional-expression>

| <unary-expression> <assignment-operator> <assignment-expression>

<assignment-operator> ::= =

| \*=

| /=

| %=

| +=

| -=

| <<=

| >>=

| &=

| ^=

| |=

<unary-operator> ::= &

| \*

| +

| -

| ~

| !

<type-name> ::= {<specifier-qualifier>}+ {<abstract-declarator>}?

<parameter-type-list> ::= <parameter-list>

| <parameter-list> , ...

<parameter-list> ::= <parameter-declaration>

| <parameter-list> , <parameter-declaration>

<parameter-declaration> ::= {<declaration-specifier>}+ <declarator>

| {<declaration-specifier>}+ <abstract-declarator>

| {<declaration-specifier>}+

<abstract-declarator> ::= <pointer>

| <pointer> <direct-abstract-declarator>

| <direct-abstract-declarator>

<direct-abstract-declarator> ::= ( <abstract-declarator> )

| {<direct-abstract-declarator>}? [ {<constant-expression>}? ]

| {<direct-abstract-declarator>}? ( {<parameter-type-list>}? )

<enum-specifier> ::= enum <identifier> { <enumerator-list> }

| enum { <enumerator-list> }

| enum <identifier>

<enumerator-list> ::= <enumerator>

| <enumerator-list> , <enumerator>

<enumerator> ::= <identifier>

| <identifier> = <constant-expression>

<typedef-name> ::= <identifier>

<declaration> ::= {<declaration-specifier>}+ {<init-declarator>}\* ;

<init-declarator> ::= <declarator>

| <declarator> = <initializer>

<initializer> ::= <assignment-expression>

| { <initializer-list> }

| { <initializer-list> , }

<initializer-list> ::= <initializer>

| <initializer-list> , <initializer>

<compound-statement> ::= { {<declaration>}\* {<statement>}\* }

<statement> ::= <labeled-statement>

| <expression-statement>

| <compound-statement>

| <selection-statement>

| <iteration-statement>

| <jump-statement>

<labeled-statement> ::= <identifier> : <statement>

| case <constant-expression> : <statement>

| default : <statement>

<expression-statement> ::= {<expression>}? ;

<selection-statement> ::= if ( <expression> ) <statement>

| if ( <expression> ) <statement> else <statement>

| switch ( <expression> ) <statement>

<iteration-statement> ::= while ( <expression> ) <statement>

| do <statement> while ( <expression> ) ;

| for ( {<expression>}? ; {<expression>}? ; {<expression>}? ) <statement>

<jump-statement> ::= goto <identifier> ;

| continue ;

| break ;

| return {<expression>}? ;

Lua

chunk ::= {stat [`;´]} [laststat[`;´]]

block ::= chunk

stat ::= varlist1 `=´ explist1 |

functioncall |

do block end |

while exp do block end |

repeat block until exp |

if exp then block {elseif exp then block} [else block] end |

for Name `=´ exp `,´ exp [`,´ exp] do block end |

for namelist in explist1 do block end |

function funcname funcbody |

local function Name funcbody |

local namelist [`=´ explist1]

laststat ::= return [explist1] | break

funcname ::= Name {`.´ Name} [`:´ Name]

varlist1 ::= var {`,´ var}

var ::= Name | prefixexp `[´ exp `]´ | prefixexp `.´ Name

namelist ::= Name {`,´ Name}

explist1 ::= {exp `,´} exp

exp ::= nil | false | true | Number | String | `...´ |

function | prefixexp | tableconstructor | exp binop exp | unop exp

prefixexp ::= var | functioncall | `(´ exp `)´

functioncall ::= prefixexp args | prefixexp `:´ Name args

args ::= `(´ [explist1] `)´ | tableconstructor | String

function ::= function funcbody

funcbody ::= `(´ [parlist1] `)´ block end

parlist1 ::= namelist [`,´ `...´] | `...´

tableconstructor ::= `{´ [fieldlist] `}´

fieldlist ::= field {fieldsep field} [fieldsep]

field ::= `[´ exp `]´ `=´ exp | Name `=´ exp | exp

fieldsep ::= `,´ | `;´

binop ::= `+´ | `-´ | `\*´ | `/´ | `^´ | `%´ | `..´ |

`<´ | `<=´ | `>´ | `>=´ | `==´ | `~=´ |

and | or

unop ::= `-´ | not | `#´