# grammar – moses0.1

// GRAMMAR OF A STATEMENT

***statement -> compound-statement***

***| if-statement***

***| while-statement***

***| break-statement***

***| continue-statement***

***| return-statement***

***| expression-statement***

***| declaration-statement***

***if-statement ->***

***“if” expression compound-statement “else” compound-statement***

***while-statement -> “while” expression compound-statement***

***break-statement -> “break” “;”***

***compound-statement -> “{” statement\* “}”***

***continue-statement -> “continue” “;”***

***return-statement -> “return” expression? “;” | return anonymous-initial ? “;”***

***expression-statement -> expression? “;”***

// GRAMMAR OF EXPRESSION

***expression -> assignment-expression***

***assignment-expression -> condition-or-expression***

***| unary-expression assignment-operator condition-or-expression***

***assignment-operator -> “=” | “\*=” | “/=” | “+=” | “-=” |”&&=” | “||=”***

***cond-or-expression -> condition-and-expression***

***| condition-or-expression “||” cond-andition-expression***

***cond-and-expression -> equality-expression***

***| condition-and-expression “&&” equality-expression***

***equality-expression -> rel-expression***

***| equality-expression “==” rel-expression***

***| equality-expression “!=” rel-expression***

***rel-expression -> additive-expression***

***| rel-expression “<” additive-expression***

***| rel-expression “<=” additive-expression***

***| rel-expression “>” additive-expression***

***| rel-expression “>=” additive-expression***

***additive-expression -> m-d-expression***

***| additive-expression “+” m-d-expression***

***| additive-expression “-” m-d-expression***

***multiplicate-expression -> u-expression***

***| multiplicate-expression “\*” u-expression***

***| multiplicate-expression “/” u-expression***

***u-expression -> “-” u-expression***

***| “!” u-expression***

***| ++ u-expression***

***| -- u-expression***

***| post-expression***

***postfix-expression -> primary-expression***

***| post-expression . identifier***

***| post-expression ++***

***| post-expression --***

***primary-expression -> identifier arg-list?***

***| “(” expression “)”***

***| INTLITERAL***

***| BOOLLITERAL***

// GRAMMAR OF PARAMETERS

***para-list -> “(” proper-para-list? ”)”***

***proper-para-list -> para-declaration ( “,” para-declaration ) \****

***para-declaration -> identifier type-annotation***

***arg-list -> “(” proper-arg-list ? “)”***

***proper-arg-list -> arg ( “,” arg ) \****

***arg -> expression | anonymous -initializer***

// GRAMMAR OF DECLARATION

***declaration-statement -> constant-declaration***

***| variable-declaration***

***| class-declaration***

***| unpack-declaration***

***function-definition -> func identifier para-list “->” return-type compound-statement***

***return-type -> type | “void” | anonymous***

***variable-declaration -> “var” identifier initializer “;”***

***| “var” identifier type-annotation “;”***

***unpack-declaration -> “var ” “{“ unpack-decl-internal “}” = upack-initial***

***unpack-initial -> identifier arg-list ?***

***unpack-decl-internal -> identifier ( “,” identifier )\****

***initializer -> “=” expression |“=” anonymous-initializer***

***anonymous -initializer -> “{” anonymous -initial-internal “}”***

***anonymous -initial-internal -> anonymous -initial-element (“,” class-initial-element)\****

***anonymous -initial-element -> expression | anonymous -initializer***

***class-declaration -> “class” identifier class-body “;”***

***class-body -> “{” ( declaration-statement | function-definition )\* “}”***

***constant-declaration -> “const” identifier init-expression “;”***

***| “const” identifier type-annotation “;”***

***init-expression -> “=” expression***

***type-annotation -> “:” type***

***anonymous -> “{” anonymous-internal “}”***

***anonymous-interal -> anonymous-type ( “,” anonymous-type)\****

***anonymous-type -> “int” | “bool” | anonymous***

// GRAMMAR OF PRIMITIVE TYPES

***type -> “int” | “bool” | identifier | anonymous***

// GRAMMAR OF IDENTIFIERS

***identifier -> ID***

// TOP-LEVEL

***top\_level : (statement | function-definition )\****

# moses 0.1 – LL(1)

*statement ->* ***compound-statement*** *|* ***if-statement*** *|* ***while-statement*** *|* ***break-statement*** *|* ***continue-statement*** *|* ***return-statement*** *|* ***expression-statement*** *|****declaration-statement***

*if-statement ->* ***if******expression******compound-statement*** *else* ***compound-statement***

*while-statement ->* ***while******expression******compound-statement***

*break-statement ->* ***break******;***

*compound-statement ->* ***{******statement-list******}***

*statement-list ->* ***EPSILON*** *|* ***statement******statement-list***

*continue-statement ->* ***continue ;***

*return-statement ->* ***return******expression******;***

*return-statement ->* ***return anonymous-initial ;***

*return-statement ->* ***return ;***

*expression-statement ->* ***expression-list******;***

*expression-list ->* ***expression*** *|* ***EPSILON***

*expression ->* ***assignment-expression***

*assignment-expression ->* ***condition-or-expression***

***| u-expression assignment-operator condition-expression***

*assignment-operator ->* ***= | \*= | /= | += | -= | &&= | XX=***

*condition-or-expression ->* ***condition-and-expression condition-or-expression-tail***

*condition-or-expression-tail ->* ***EPSILON*** *|* ***XX******condition-and-expression condition-or-expression-tail***

*condition-and-expression ->* ***equality-expression******condition-and-expression-tail***

*condition-and-expression-tail ->* ***&&******equality-expression equality-expression-tail*** *|* ***EPSILON***

*equality-expression ->* ***rel-expression******equality-expression-tail***

*equality-expression-tail ->* ***EPSILON*** *|* ***==******rel-expression equality-expression-tail*** *|* ***!=******rel-expression equality-expression-tail***

*rel-expression ->* ***additive-expression rel-expression-tail***

*rel-expression-tail ->* ***EPSILON*** *|* ***<******additive-expression rel-expression-tail*** *|* ***<=******additive-expression******rel-expression-tail*** *|* ***>******additive-expression rel-expression-tail*** *|* ***>=******additive-expression rel-expression-tail***

*additive-expression ->* ***m-d-expression additive-expression-tail***

*additive-expression-tail ->* ***EPSILON*** *|* ***+******m-d-expression additive-expression-tail*** *|* ***-******m-d-expression additive-expression-tail***

*m-d-expression ->* ***u-expression m-d-expression-tail***

*m-d-expression-tail ->* ***EPSILON*** *|* ***\*******u-expression m-d-expression-tail*** *|* ***/******u-expression m-d-expression-tail***

*u-expression ->* ***-******u-expression*** *|* ***!******u-expression*** *|* ***++ u-expression | -- u-expression***

***| post-expression***

*post-expression ->* ***primary-expression | primary-expression post-expression-tail***

*post-expression-tail ->* ***. identifier post-expression-tail | ++ post-expression-tail***

***| -- post-expression-tail | EPSILON***

*primary-expression ->* ***identifier*** *|* ***identifier arg-list*** *|* ***(******expression******)*** *|* ***INT-LITERAL*** *|* ***BOOL-LITERAL***

*para-list ->* ***( )*** *|* ***(******proper-para-list******)***

*proper-para-list ->* ***para-declaration proper-para-list-tail***

*proper-para-list-tail ->* ***,******para-declaration proper-para-list-tail*** *|* ***EPSILON***

*para-declaration ->* ***identifier type-annotation***

*arg-list ->* ***( )*** *| (* ***proper-arg-list*** *)*

*proper-arg-list ->* ***arg proper-arg-list-tail***

*proper-arg-list-tail ->* ***,******arg proper-arg-list-tail*** *|* ***EPSILON***

*arg ->* ***expression | anonymous -initial***

*declaration-statement ->* ***constant-declaration*** *|* ***variable-declaration*** *|* ***class-declaration | unpack-declaration***

*function-definition ->* ***func******identifier******para-list -> return-type compound-statement***

*return-type ->* ***type | void***

*variable-declaration ->* ***var******identifier******initial******;*** *|* ***var identifier type-annotation ;***

*unpack-declaration ->* ***var unpack-decls = unpack-initial ;***

*unpack-initial ->* ***identifier | identifier arg-list***

*unpack-decls ->* ***{ unpack-decl-internal }***

*unpack-decl-internal ->* ***unpack-element unpack-decl-internal-tail***

*unpack-decl-internal-tail ->* ***, unpack-element unpack-decl-internal-tail | EPSILON***

*unpack-element ->* ***identifier | unpack-decls***

*class-declaration ->* ***class******identifier******class-body ;***

*class-body ->* ***{ class-member******}***

*class-member ->* ***declaration-statement class-member*** *|* ***function-definition class-member***

*|* ***EPSILON***

*constant-declaration ->* ***const******identifier******init-expression******;*** *|* ***const******identifier******type-annotation ;***

*initial ->* ***= expression | = anonymous-initial***

*anonymous-initial ->* ***{ anonymous-initial-internal }***

*anonymous-initial-internal ->* ***anonymous-initial-element anonymous-initial-internal-tail***

*anonymous-initial-internal-tail ->* ***, anonymous-initial-element anonymous-initial-internal-tail | EPSILON***

*anonymous-initial-element ->* ***expression | anonymous-initial***

*type-annotation ->* ***:******type***

*anonymous ->* ***{ anonymous-annotation-internal }***

*anonymous-internal ->* ***anonymous-type anonymous-internal-tail***

*anonymous-internal-tail ->* ***, anonymous-type anonymous-internal-tail | EPSILON***

*anonymous-type ->* ***int | bool | anonymous***

*type ->* ***int*** *|* ***bool | identifier | anonymous***

*top-level ->* ***statement******top-level*** *|* ***function-definition top-level | EPSILON***

***(注：由于‘||’运算会被识别为分隔符，所以使用XX代替)***