

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?** **";"**

expression-statement -> **expression?** **";"**

class-body -> **"{"** **variable-declaration*** **"}"**

// GRAMMAR OF EXPRESSION

expression -> **assignment-expression**

assignment-expression -> **condition-or-expression** **"="** **condition-or-expression**

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

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

multiply-expression -> **u-expression**
| multiply-expression "*" u-expression
| multiply-expression "/" u-expression

unary-expression -> "-" unary-expression
| "!" unary-expression
| primary-expression

primary-expression -> **identifier arg-list?**
| "(" expression ")"
| INTLITERAL
| BOOLLITERAL

// GRAMMAR OF PARAMETERS

para-list -> "(" proper-para-list? ")"
proper-para-list -> **para-declaration** ("," para-decl) *
para-decl -> **type identifier**
arg-list -> "(" proper-arg-list ? ")"
proper-arg-list -> **arg** ("," arg) *
arg -> **expression**

// GRAMMAR OF DECLARATION

declaration-statement -> **function-declaration**
| **constant-declaration**
| **variable-declaration**
| **class-declaration**
function-declaration -> **identifier para-list compound-statement**
variable-declaration -> "**var**" identifier initializer ";"
| "**var**" identifier type-annotation ";"

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

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

init-expression -> "=" expression
type-annotation -> ":" type

```
// GRAMMAR OF PRIMITIVE TYPES
```

```
type -> "int" | "bool"
```

```
// GRAMMAR OF IDENTIFIERS
```

```
identifier -> ID
```

```
// TOP-LEVEL
```

```
top_level : statement*
```

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** ;

expression-statement -> *expression-list* ;

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

class-body -> { *variable-declaration-list* }

variable-declaration-list -> *variable-declaration* *variable-declaration-list* | **EPSILON**

expression -> *assignment-expression*

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

condition-or-expression-list -> *condition-or-expression* = *condition-or-expression-list* | **EPSILON**

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

condition-or-expression-tail -> **EPSILON** | **||** *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* | *primary-expression*

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 -> **type** *identifier*

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*

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

function-declaration -> **identifier** *para-list* *compound-statement*

variable-declaration -> **var identifier** *init-expression* ; | **var identifier** *type-annotation* ;

class-declaration -> **class identifier** *init-expression* ; | **class identifier** *type-annotation* ;

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

init-expression -> = *expression*

type-annotation -> : *type*

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

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