Mint Programming Language — Grammar Design | MILESTONE - 2

Team Name: SER502-Group25
Language Name: Mint
Milestone: 2 — Grammar Design and Tokenization
Course: SER 502 — Language and Programming Paradigms
Semester: Spring 2025

1. Introduction

Mint is a beginner-friendly imperative programming language developed as part of our SER 502 course project. The language takes inspiration from modern C-style languages and is designed to be simple, readable, and cross-platform.
In this milestone, our focus was to define the grammar of the language and ensure it can be tokenized and parsed successfully using ANTLR4. This document includes the language's lexical structure and parser grammar written in EBNF.

2. Language Features Covered

- Variable declarations with types:
  - mint_int, mint_float, mint_string, mint_bool
- Arithmetic expressions:
  - +, -, *, /, %
- Logical operators:
  - and, or, not
- Relational comparisons:
  - ==, !=, <, <=, >, >=
- Conditional control flow:
  - mint_if, mint_else, mint_elseif
- Iteration constructs:
  - mint_while, mint_for
- Output commands:
  - say(), sayln()
- Ternary operator:
  - ? :
- Flow control:
  - mint_break, mint_continue

3. Token Definitions (Lexical Rules)

Tokens are defined using regular expressions as follows:

```
// ====================
// Lexer Rules
// ====================

MINT_IF       : 'mint_if';
```

```
MINT_ELSE     : 'mint_else';
MINT_ELSEIF   : 'mint_elseif';
MINT_FOR      : 'mint_for';
MINT_WHILE    : 'mint_while';
MINT_BREAK    : 'mint_break';
MINT_CONTINUE : 'mint_continue';
SAY           : 'say';
SAYLN         : 'sayln';
T_IF          : '?';
T_ELSE        : ':';

AND           : 'and';
OR            : 'or';
NOT           : 'not';

ADD           : '+';
SUB           : '-';
MUL           : '*';
DIV           : '/';
MOD           : '%';

EQ            : '==';
NEQ           : '!=';
LT            : '<';
LTE           : '<=';
GT            : '>';
GTE           : '>=';

ASSIGN        : '=';
SEMI          : ';';
COMMA         : ',';
LPAREN        : '(';
RPAREN        : ')';
LBRACE        : '{';
RBRACE        : '}';

INT_TYPE      : 'mint_int';
FLOAT_TYPE    : 'mint_float';
STRING_TYPE   : 'mint_string';
BOOL_TYPE     : 'mint_bool';

BOOL          : 'true' | 'false';
NUMBER        : [0-9]+('.'[0-9]+)?;
STRING        : '"' .*? '"';
```

```
IDENTIFIER    : [a-zA-Z_][a-zA-Z_0-9]*;

WS            : [ \t\r\n]+ -> skip;
LINE_COMMENT  : '//' ~[\r\n]* -> skip;
```

4. Grammar Rules (EBNF)

```
// =====================
// Parser Rules
// =====================
program
: statement* EOF
;
statement
: declaration
| assignment
| printStatement
| ifStatement
| whileLoop
| forLoop
| breakStatement
| continueStatement
| expressionStatement
;
declaration
: type IDENTIFIER (ASSIGN expression)? SEMI
;
assignment
: IDENTIFIER ASSIGN expression SEMI
;
printStatement
: SAY LPAREN expression RPAREN SEMI
| SAYLN LPAREN expression RPAREN SEMI
;
ifStatement
: MINT_IF LPAREN expression RPAREN block
(MINT_ELSEIF LPAREN expression RPAREN block)*
(MINT_ELSE block)?
;
whileLoop
: MINT_WHILE LPAREN expression RPAREN block
;
simpleAssignment
: IDENTIFIER ASSIGN expression
;
```

```
forLoop
: MINT_FOR LPAREN simpleAssignment SEMI expression SEMI
simpleAssignment RPAREN block
;
breakStatement
: MINT_BREAK SEMI
;
continueStatement
: MINT_CONTINUE SEMI
;
expressionStatement
: expression SEMI
;
block
: LBRACE statement* RBRACE
;
type
: INT_TYPE
| FLOAT_TYPE
| STRING_TYPE
| BOOL_TYPE
;
// =====================
// Expression Parsing by Precedence
// =====================
expression
: ternaryExpression
;
ternaryExpression
: logicalExpression (T_IF expression T_ELSE expression)?
;
logicalExpression
: logicalExpression AND equalityExpression
| logicalExpression OR equalityExpression
| NOT logicalExpression
| equalityExpression
;
equalityExpression
: comparisonExpression ((EQ | NEQ) comparisonExpression)*
;
comparisonExpression
: additiveExpression ((LT | LTE | GT | GTE) additiveExpression)*
;
additiveExpression
```

```
: additiveExpression (ADD | SUB) multiplicativeExpression
| multiplicativeExpression
;
multiplicativeExpression
: multiplicativeExpression (MUL | DIV | MOD) primaryExpression
| primaryExpression
;
primaryExpression
: LPAREN expression RPAREN
| IDENTIFIER
| NUMBER
| STRING
| BOOL
;
```

5. Sample output and parse tree for Sample1.mint program:

Tree:

```
monisha@Monishas-MacBook-Air ser502-group25 % java -cp "build:antlr-4.13.2-complete.jar" org.antlr.v4.gui.TestRig gen.Mint program -tree data/sample1.mint
(program (statement (declaration (type mint_int) x = (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression 10)))))))) ;)) (statement (declaration (type mint_int) y = (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression 5)))))))) ;)) (statement (printStatement sayln ( (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (additiveExpression (multiplicativeExpression (primaryExpression x))) + (multiplicativeExpression (primaryExpression y)))))))) ) ;)) (statement (ifStatement mint_if ( (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression x))) > (additiveExpression (multiplicativeExpression (primaryExpression y)))))))) ) (block { (statement (printStatement sayln ( (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression "X is greater")))))))) ) ;)) }) mint_else (block { (statement (printStatement sayln ( (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression "Y is greater or equal")))))))) ) ;)) }))) (statement (declaration (type mint_int) i = (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression 0)))))))) ;)) (statement (forLoop mint_for ( (simpleAssignment i = (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression 0))))))))) ; (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression
```

(additiveExpression (multiplicativeExpression (primaryExpression i))) < (additiveExpression (multiplicativeExpression (primaryExpression 3)))))))) ; (simpleAssignment i = (expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (additiveExpression (multiplicativeExpression (primaryExpression i))) + (multiplicativeExpression (primaryExpression 1))))))))) (block { (statement (printStatement say ((expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression "loop: "))))))))) ;)) (statement (printStatement sayln ((expression (ternaryExpression (logicalExpression (equalityExpression (comparisonExpression (additiveExpression (multiplicativeExpression (primaryExpression i))))))))) ;)) })))) <EOF>)

Token:
monisha@Monishas-MacBook-Air ser502-group25 % java -cp "build:antlr-4.13.2-complete.jar" org.antlr.v4.gui.TestRig gen.Mint program -tokens data/sample1.mint
[@0,0:7='mint_int',<'mint_int'>,1:0]
[@1,9:9='x',<IDENTIFIER>,1:9]
[@2,11:11='=',<'='>,1:11]
[@3,13:14='10',<NUMBER>,1:13]
[@4,15:15=';',<';'>,1:15]
[@5,17:24='mint_int',<'mint_int'>,2:0]
[@6,26:26='y',<IDENTIFIER>,2:9]
[@7,28:28='=',<'='>,2:11]
[@8,30:30='5',<NUMBER>,2:13]
[@9,31:31=';',<';'>,2:14]
[@10,34:38='sayln',<'sayln'>,4:0]
[@11,39:39='(',<'('>,4:5]
[@12,40:40='x',<IDENTIFIER>,4:6]
[@13,42:42='+',<'+'>,4:8]
[@14,44:44='y',<IDENTIFIER>,4:10]
[@15,45:45=')',<')'>,4:11]
[@16,46:46=';',<';'>,4:12]
[@17,49:55='mint_if',<'mint_if'>,6:0]
[@18,57:57='(',<'('>,6:8]
[@19,58:58='x',<IDENTIFIER>,6:9]
[@20,60:60='>',<'>'>,6:11]
[@21,62:62='y',<IDENTIFIER>,6:13]
[@22,63:63=')',<')'>,6:14]
[@23,65:65='{',<'{'>,6:16]
[@24,71:75='sayln',<'sayln'>,7:4]
[@25,76:76='(',<'('>,7:9]
[@26,77:90='"X is greater"',<STRING>,7:10]
[@27,91:91=')',<')'>,7:24]
[@28,92:92=';',<';'>,7:25]
[@29,94:94='}',<'}'>,8:0]

[@30,96:104='mint_else',<'mint_else'>,8:2]
[@31,106:106='{',<'{'>,8:12]
[@32,112:116='sayln',<'sayln'>,9:4]
[@33,117:117='(',<'('>,9:9]
[@34,118:140='"Y is greater or equal"',<STRING>,9:10]
[@35,141:141=')',<')'>,9:33]
[@36,142:142=';',<';'>,9:34]
[@37,144:144='}',<'}'>,10:0]
[@38,147:154='mint_int',<'mint_int'>,12:0]
[@39,156:156='i',<IDENTIFIER>,12:9]
[@40,158:158='=',<'='>,12:11]
[@41,160:160='0',<NUMBER>,12:13]
[@42,161:161=';',<';'>,12:14]
[@43,163:170='mint_for',<'mint_for'>,13:0]
[@44,172:172='(',<'('>,13:9]
[@45,173:173='i',<IDENTIFIER>,13:10]
[@46,175:175='=',<'='>,13:12]
[@47,177:177='0',<NUMBER>,13:14]
[@48,178:178=';',<';'>,13:15]
[@49,180:180='i',<IDENTIFIER>,13:17]
[@50,182:182='<',<'<'>,13:19]
[@51,184:184='3',<NUMBER>,13:21]
[@52,185:185=';',<';'>,13:22]
[@53,187:187='i',<IDENTIFIER>,13:24]
[@54,189:189='=',<'='>,13:26]
[@55,191:191='i',<IDENTIFIER>,13:28]
[@56,193:193='+',<'+'>,13:30]
[@57,195:195='1',<NUMBER>,13:32]
[@58,196:196=')',<')'>,13:33]
[@59,198:198='{',<'{'>,13:35]
[@60,204:206='say',<'say'>,14:4]
[@61,207:207='(',<'('>,14:7]
[@62,208:215='"loop: "',<STRING>,14:8]
[@63,216:216=')',<')'>,14:16]
[@64,217:217=';',<';'>,14:17]
[@65,223:227='sayln',<'sayln'>,15:4]
[@66,228:228='(',<'('>,15:9]
[@67,229:229='i',<IDENTIFIER>,15:10]
[@68,230:230=')',<')'>,15:11]
[@69,231:231=';',<';'>,15:12]
[@70,233:233='}',<'}'>,16:0]
[@71,234:233='<EOF>',<EOF>,16:1]