Mint Programming Language

SER 502-Team 25

Spring 2025

Contributors:

1. Kiran Venkatachalam (1232216299)
2. Monisha Krishnamurthy (1233351056)
3. Rahul Ravindra Reddy (1230077448)
4. Vishnu Kumar Adhilakshmi Kalidas (1234244234)

Copyright © 2023 Arizona Board of Regents

Overview

1. Introduction
2. Grammar & Tokens
3. Evaluator & Interpreter Design
4. Sample Code Execution
5. Future Scope

Copyright © 2023 Arizona Board of Regents

1.

INTRODUCTION

Copyright © 2023 Arizona Board of Regents

1. **Mint** is a beginner-friendly, imperative programming language designed to provide a smooth learning curve for new programmers while offering a robust foundation for structured programming.

2. **Cross-platform compatibility**: Mint is implemented in Java, enabling it to run seamlessly on Windows, macOS, and Linux systems. Its runtime behavior is consistent across platforms, making it a reliable tool for developers regardless of their OS.

3. **Full interpretation pipeline**: Mint supports the complete interpretation process—from **lexical analysis** (tokenizing source code) to **parsing** (syntax validation) and **runtime execution** (evaluating code through a visitor-based interpreter).

4. **ANTLR4-based architecture**: The language's grammar is defined using **ANTLR 4.13.2**, a powerful parser generator. This enables us to convert source code into parse trees efficiently, facilitating both syntax checking and semantic processing.

5. **Readable, modern syntax**: Inspired by modern C-style imperative languages, Mint provides familiar constructs such as loops, conditionals, arithmetic expressions, and more. Keywords are chosen for clarity (e.g., mint_if, mint_for, sayln) to enhance readability and reduce the cognitive load on beginners.

5. **Educational and extensible**: Mint is designed with educational goals in mind. It offers strong type checking and clearly defined semantics, making it suitable for teaching compiler and interpreter design. Additionally, its modular structure allows for future extensions, such as support for functions, data structures, and concurrency.

Copyright © 2023 Arizona Board of Regents

2.

GRAMMAR & TOKENS

Copyright © 2023 Arizona Board of Regents

GRAMMAR – Mint.g4

Mint.g4

```
// File: Mint.g4
// Author: Kiran Venkatachalam, Monisha Krishnamurthy, Rahul Ravindra Reddy, Vishnu Kumar Adhilakshmi Kalidas
// Purpose: Grammar definition for Mint language parser using ANTLR4
// Version: 5.0

grammar Mint;

@header {
package gen;
}

// ====================
// 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        : '}';
```

Mint.g4

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

NUMBER        : [0-9]+('.'[0-9]+)?;
BOOL          : 'true' | 'false';
STRING        : '"' .*? '"';
IDENTIFIER    : [a-zA-Z_][a-zA-Z_0-9]*;

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

// ====================
// 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)?
    ;
```

Copyright © 2023 Arizona Board of Regents

Mint.g4

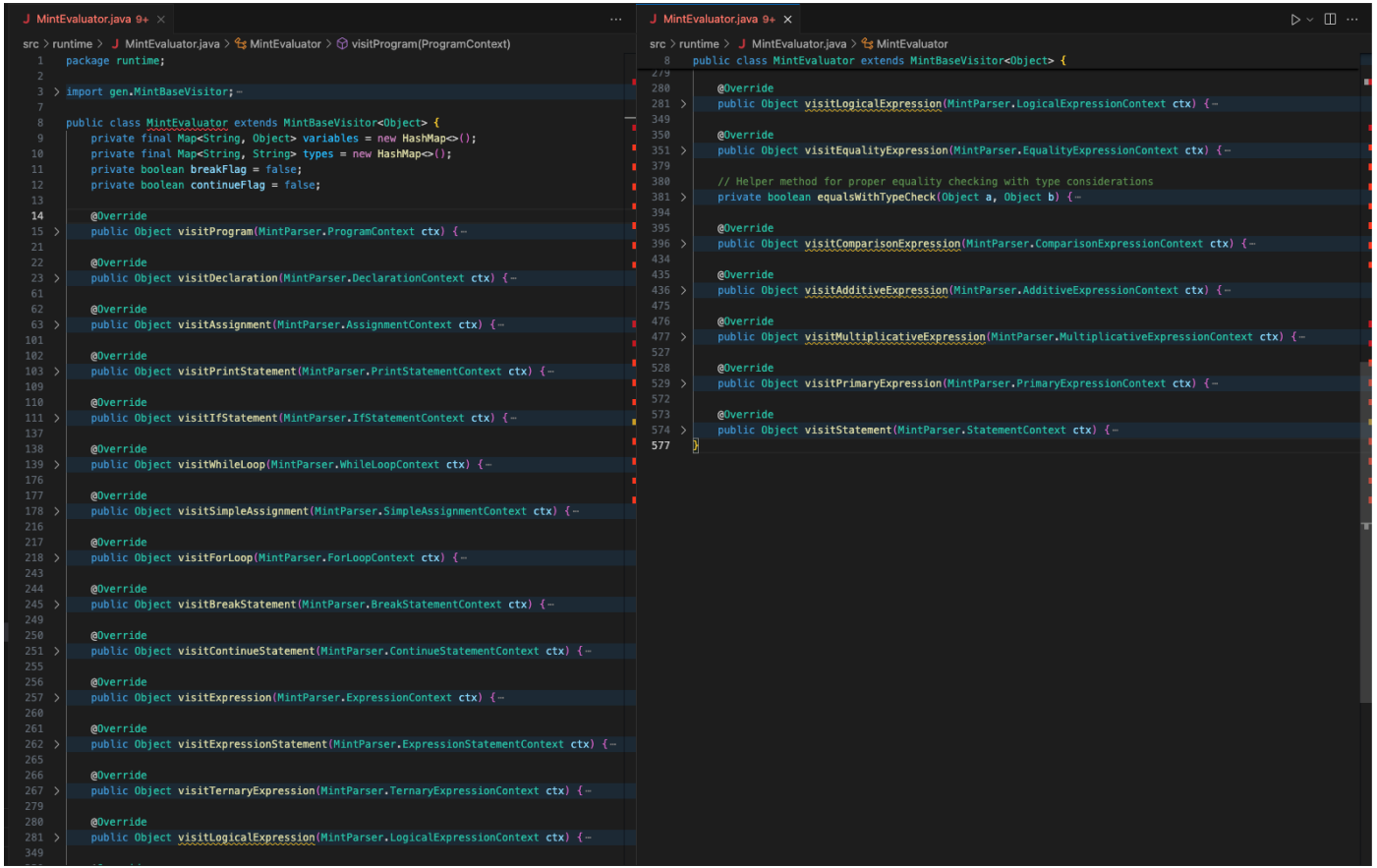
```
105  whileLoop
106      : MINT_WHILE LPAREN expression RPAREN block
107      ;
108
109  simpleAssignment
110      : IDENTIFIER ASSIGN expression
111      ;
112
113  forLoop
114      : MINT_FOR LPAREN simpleAssignment SEMI expression SEMI simpleAssignment RPAREN block
115      ;
116
117  breakStatement
118      : MINT_BREAK SEMI
119      ;
120
121  continueStatement
122      : MINT_CONTINUE SEMI
123      ;
124
125  expressionStatement
126      : expression SEMI
127      ;
128
129  block
130      : LBRACE statement* RBRACE
131      ;
132
133  type
134      : INT_TYPE
135      | FLOAT_TYPE
136      | STRING_TYPE
137      | BOOL_TYPE
138      ;
139
140  // ====================
141  // Expression Parsing by Precedence
142  // ====================
143
144  expression
145      : ternaryExpression
146      ;
147
148  ternaryExpression
149      : logicalExpression (T_IF expression T_ELSE expression)?
150      ;
151
```

Mint.g4

Mint.g4

```
152  logicalExpression
153      : logicalExpression AND equalityExpression
154      | logicalExpression OR equalityExpression
155      | NOT logicalExpression
156      | equalityExpression
157      ;
158
159  equalityExpression
160      : comparisonExpression ((EQ | NEQ) comparisonExpression)*
161      ;
162
163  comparisonExpression
164      : additiveExpression ((LT | LTE | GT | GTE) additiveExpression)*
165      ;
166
167  additiveExpression
168      : additiveExpression (ADD | SUB) multiplicativeExpression
169      | multiplicativeExpression
170      ;
171
172  multiplicativeExpression
173      : multiplicativeExpression (MUL | DIV | MOD) primaryExpression
174      | primaryExpression
175      ;
176
177  primaryExpression
178      : LPAREN expression RPAREN
179      | IDENTIFIER
180      | NUMBER
181      | STRING
182      | BOOL
183      ;
184
```

Mint Evaluator Methods Implemented



Features

1. Control Structures:

- mint_if, mint_else, mint_elseif – for conditional branching
- mint_while, mint_for – for iteration
- mint_break, mint_continue – for flow control inside loops

2. Print Statements:

- say() – prints output without a newline
- sayln() – prints output followed by a newline

3. Arithmetic Operations:

- Addition: add_color
- Subtraction: subtract_color
- Multiplication: multiply_color
- Division: divide_color
- Modulus: modulo_color

Copyright © 2023 Arizona Board of Regents

4. Data Types:

- mint_int: Integer values
- mint_float: Floating-point values
- mint_bool: Boolean values (true, false)
- mint_string: String values (double-quoted)

5. Ternary Operator:

- ? : – Used for inline conditional expressions

6. Variable Naming Rules :

- Must begin with a letter or underscore
- Can include letters, digits, and underscores
- Examples: var1, _count, temp_val

Copyright © 2023 Arizona Board of Regents

Lexer Rules:

- Defined using regular expressions
- Reserved keywords: mint_if, mint_else, say, sayln, etc.
- Operators: +, -, *, /, %, ==, !=, <, <=, >
- Data types: mint_int, mint_float, mint_string, mint_bool
- Tokens for: identifiers, numbers, booleans, strings, whitespace, comments

Sample Lexer Snippet:

```
MINT_IF       : 'mint_if';
MINT_ELSE     : 'mint_else';
MINT_FOR      : 'mint_for';
ADD           : '+';
SUB           : '-';
NUMBER        : [0-9]+('.'[0-9]+)?;
IDENTIFIER    : [a-zA-Z_][a-zA-Z_0-9]*;
WS            : [
]+ -> skip;
```

Copyright © 2023 Arizona Board of Regents

Parser Rules Overview:

- program rule accepts multiple statements
- Supported statements: declarations, assignments, print, if, for, while, break, continue
- Operator precedence is respected in expressions: ternary → logical → equality → comparison → additive → multiplicative → primary

Sample Parser Snippet:

```
    program
   : statement* EOF;

statement
   : declaration
   | assignment
   | printStatement
   | ifStatement
   | forLoop
   | whileLoop
   | breakStatement
   | continueStatement
   | expressionStatement;

declaration
   : type IDENTIFIER (ASSIGN expression)? SEMI;
```

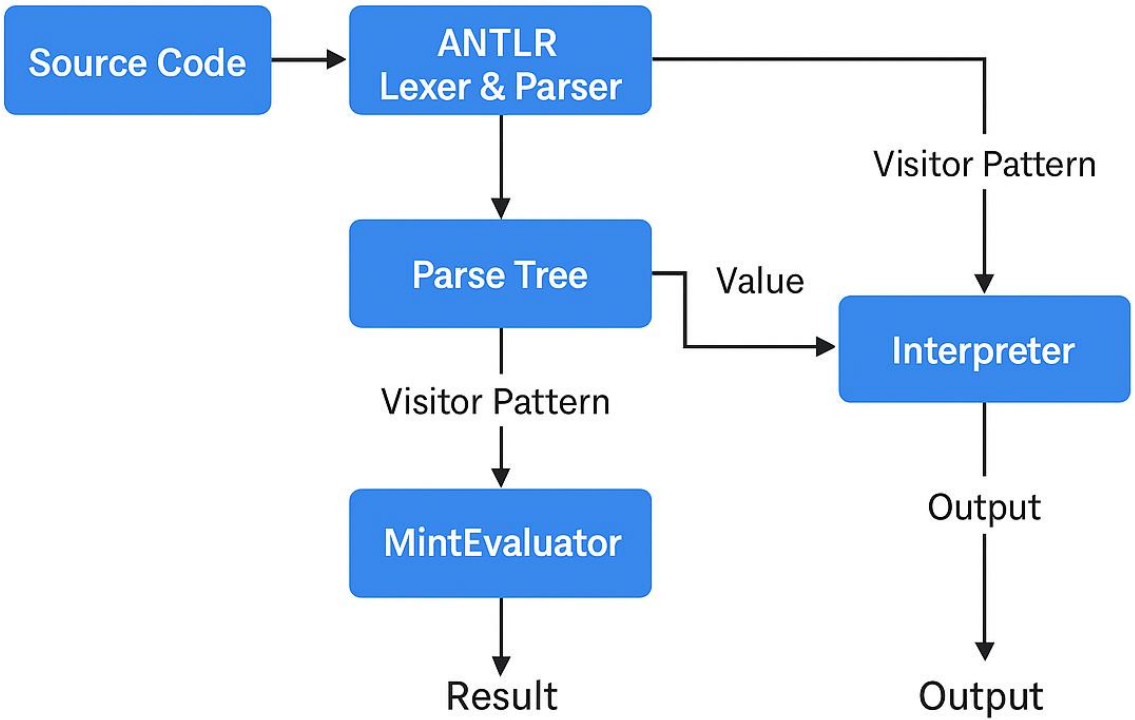
Copyright © 2023 Arizona Board of Regents

3.

EVALUATOR & INTERPRETER DESIGN

Copyright © 2023 Arizona Board of Regents

Evaluator & Interpreter Workflow



Board of Regents

4.

SAMPLE CODE EXECUTION

Copyright © 2023 Arizona Board of Regents

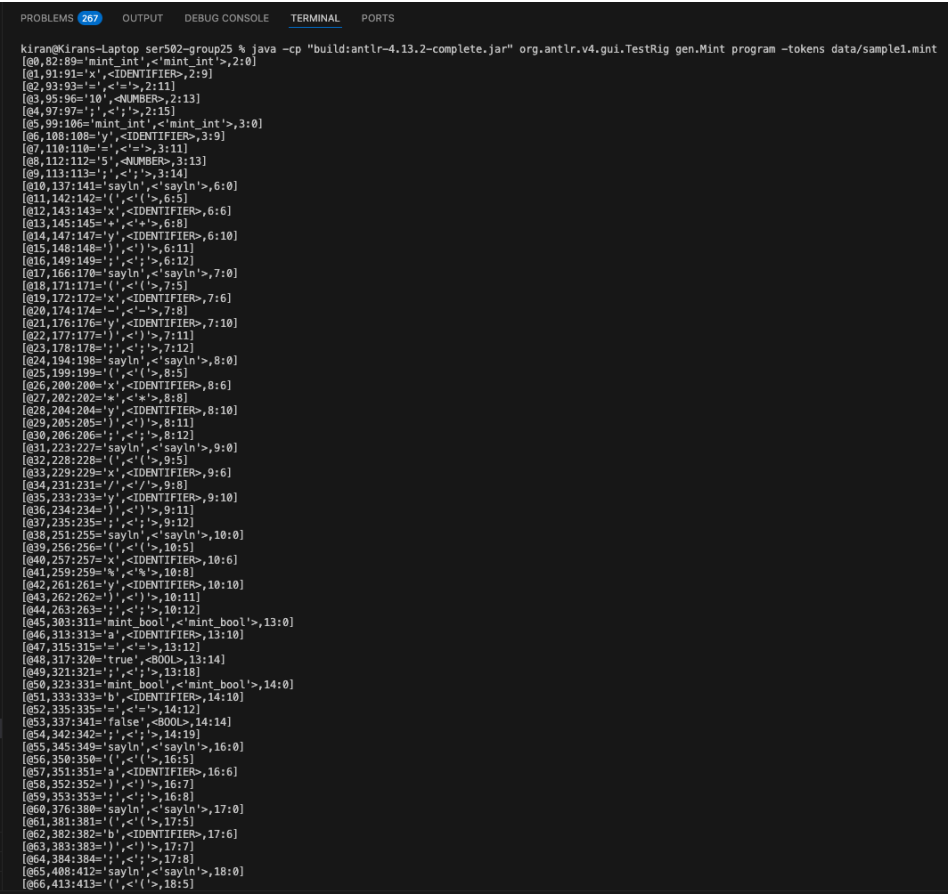
sample1.mint

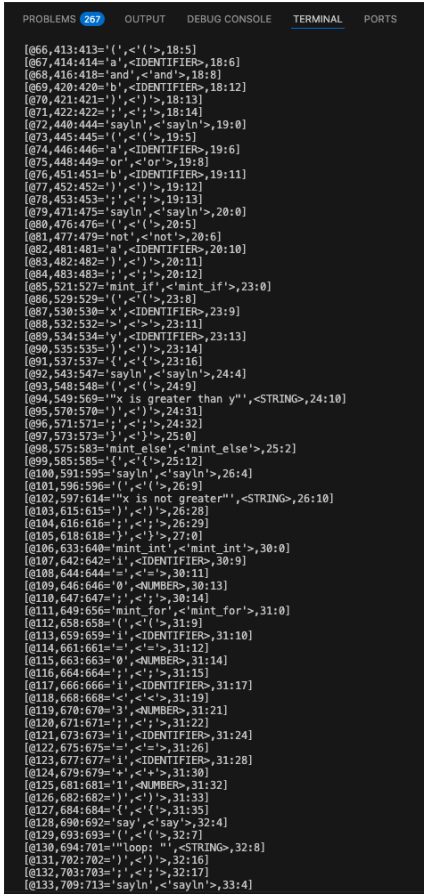
sample1.mint M

data > sample1.mint

```
// Demonstrates arithmetic, logical expressions, if-else, and for loop constructs
mint_int x = 10;
mint_int y = 5;

// Simple arithmetic
sayln(x + y);          // 15
sayln(x - y);          // 5
sayln(x * y);          // 50
sayln(x / y);          // 2
sayln(x % y);          // 0

// Logical expressions
mint_bool a = true;
mint_bool b = false;

sayln(a);              // true
sayln(b);              // false
sayln(a and b);        // false
sayln(a or b);         // true
sayln(not a);          // false

// If-else block
mint_if (x > y) {
    sayln("x is greater than y");
} mint_else {
    sayln("x is not greater");
}

// For loop
mint_int i = 0;
mint_for (i = 0; i < 3; i = i + 1) {
    say("loop: ");
    sayln(i);          // 0, 1, 2
}
```

Output – Tokens

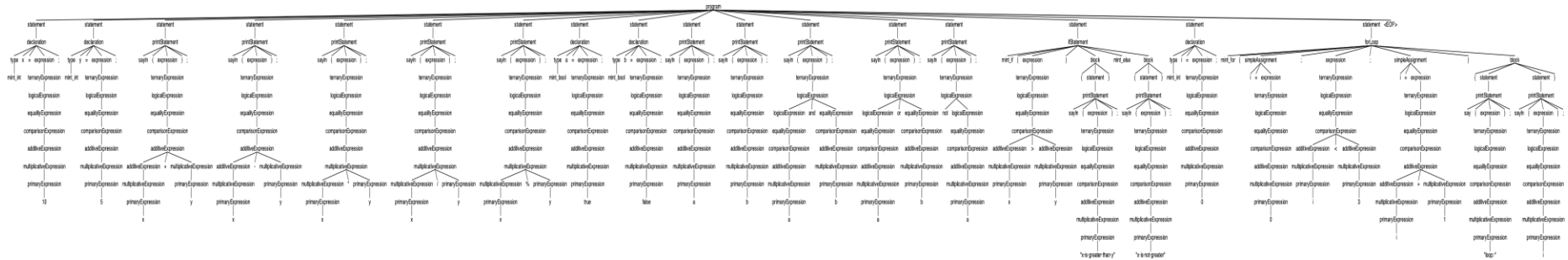




Output - Parse Tree

Output – Execution of sample1.mint

VISUAL TREE


program
statement
declaration
type x = expression ;
mint_int
ternaryExpression
logicalExpression
equalityExpression
comparisonExpression
additiveExpression
multiplicativeExpression
primaryExpression
10
type y = expression ;
mint_int
5
printStatement
sayln (expression) ;
x + y
x
y
x - y
x * y
x / y
x % y
mint_bool
true
false
a
b
logicalExpression and equalityExpression
a and b
logicalExpression or equalityExpression
a or b
not logicalExpression
not a
ifStatement
mint_if (expression) block mint_else block
x > y
"x is greater than y"
mint_int ternaryExpression
0
mint_for (simpleAssignment ; expression ; simpleAssignment) block
i = expression
i < 3
i = i + 1
i
1
text_exp <EOF>
"loop:"

5.

FUTURE SCOPE

Copyright © 2023 Arizona Board of Regents

FUTURE SCOPE:

- **Enhanced Learning Tools:**
  We aim to integrate interactive feedback for syntax errors and semantic issues directly in the console output, helping learners identify and understand mistakes intuitively.

- **Mint Playground (Web IDE):**
  A lightweight browser-based IDE for Mint will allow users to write, run, and share Mint programs without local setup. This will make the language more accessible for students and educators.

- **Domain-Specific Extensions:**
  Mint will offer optional modules or syntactic sugar tailored for specific domains, such as basic data analysis (e.g., tabular operations) or educational simulations (e.g., logic circuits), expanding Mint's use cases.

- **Standard Library Expansion:**
  A minimal standard library will be introduced to support string manipulation, math utilities, and file I/O—making it easier to write real-world programs in Mint.

- **Visual Debugger:**
  An optional debug mode with step-by-step execution and variable tracking is planned to help learners visualize program flow and state changes.

Copyright © 2023 Arizona Board of Regents

THANK YOU

Arizona State University

Copyright © 2023 Arizona Board of Regents