A Hanami Language Reference

A.1 Keyword Mapping

Table 1: Core Hanami keywords and their C++ equivalents

Hanami	C++ Equivalent	Meaning
`garden <Name>`	`namespace <Name>...`	Declares a namespace
`species <Name>`	`class <Name>...`	Declares a class
`open:`	`public:`	Public section specifier
`hidden:`	`private:`	Private section specifier
`grow <f>(...) -> <t>`	`<t> <f>(...)`	Declares a function
`bloom « x;`	`std::cout « x;`	Console output
`water » x;`	`std::cin » x;`	Console input
`branch (cond)...`	`if (cond)...`	If statement
`else branch (cond)`	`else if (cond)`	Else-if clause
`else ...`	`else ...`	Else clause

B UML Diagrams

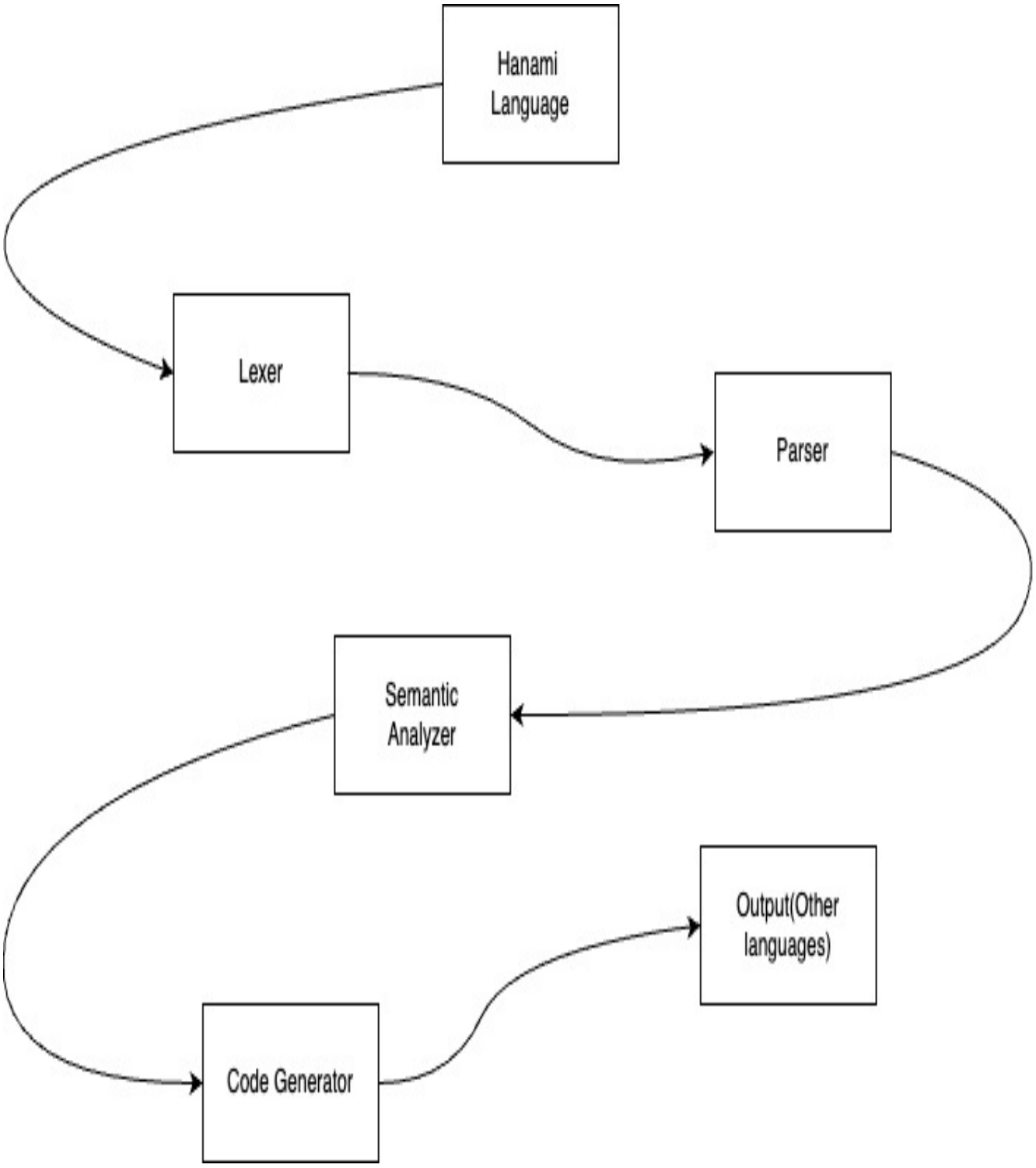
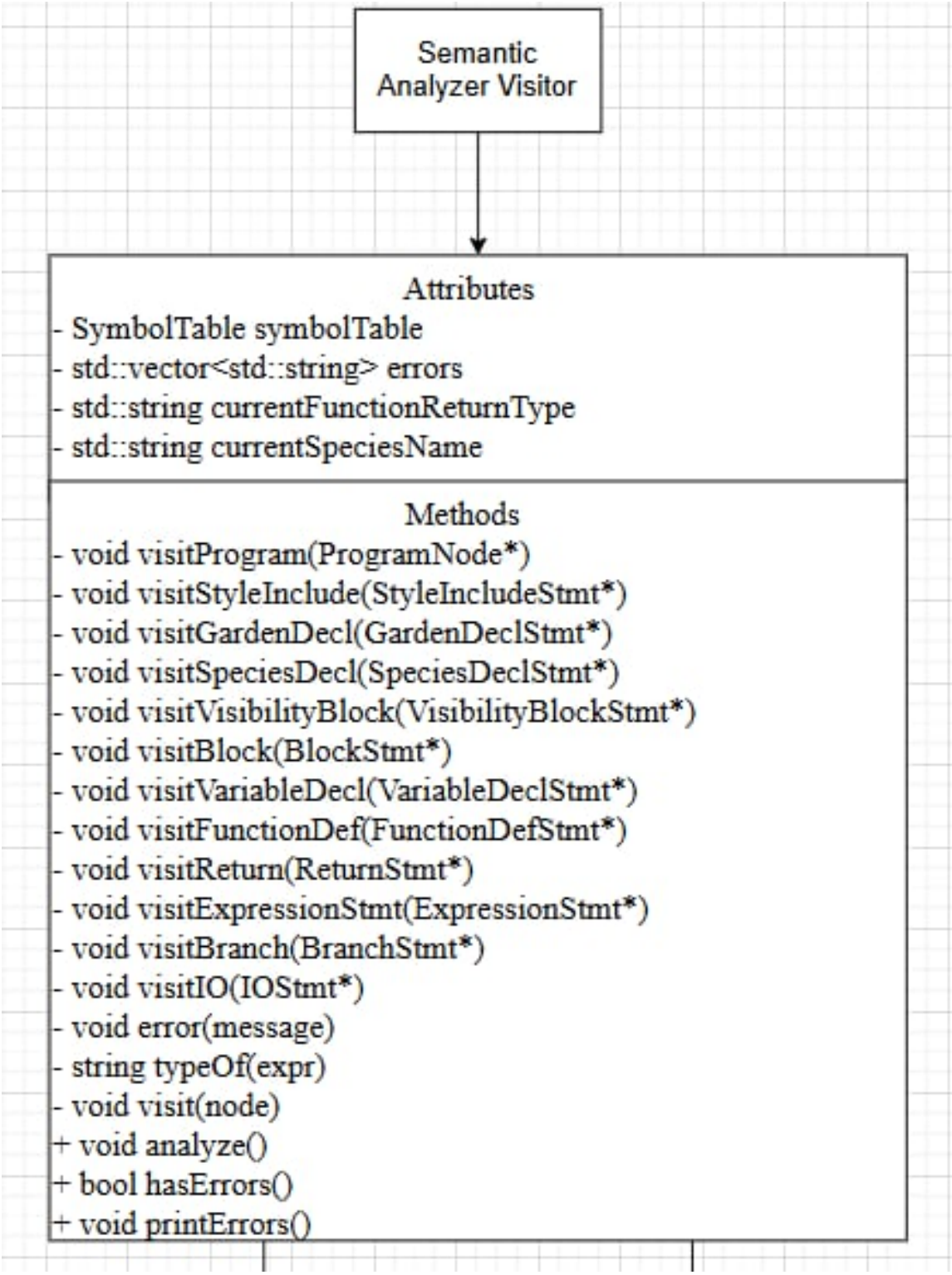
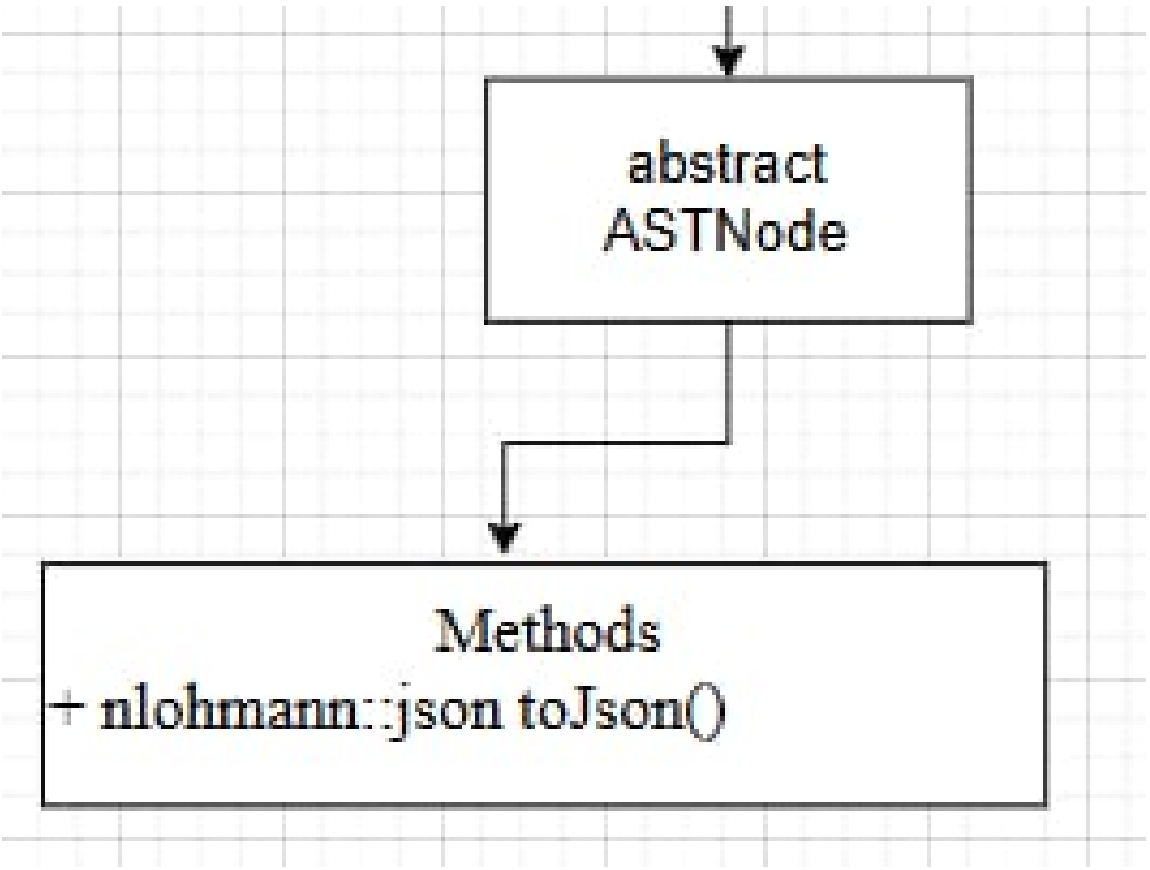
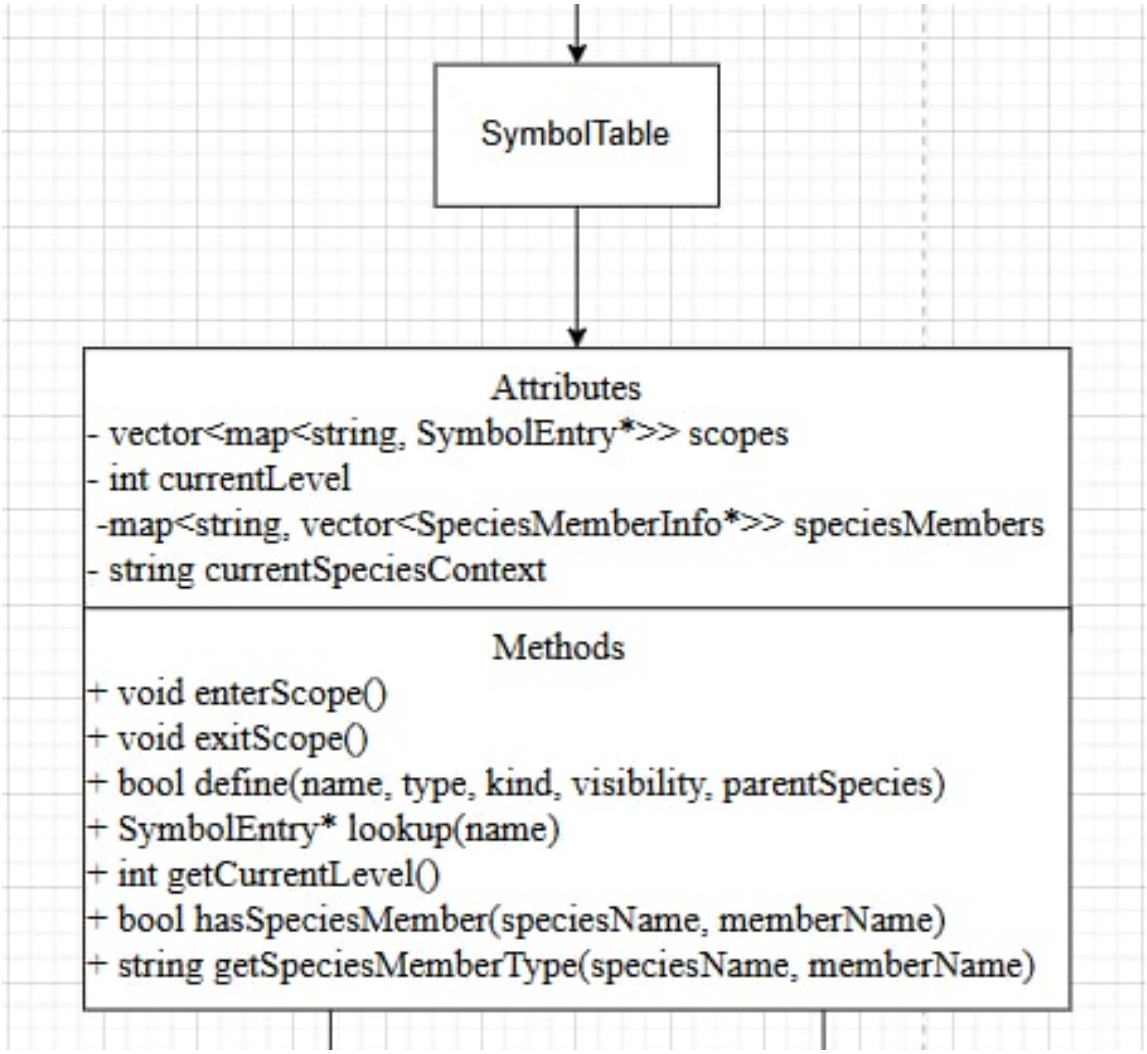


Figure 1: High-level flow from Hanami Language through Lexer, Parser, Semantic Analyzer, to Code Generator and target outputs.

C Semantic Analyzer Visitor


Semantic Analyzer Visitor
Attributes
- SymbolTable symbolTable
- std::vector<std::string> errors
- std::string currentFunctionReturnType
- std::string currentSpeciesName
Methods
- void visitProgram(ProgramNode*)
- void visitStyleInclude(StyleIncludeStmt*)
- void visitGardenDecl(GardenDeclStmt*)
- void visitSpeciesDecl(SpeciesDeclStmt*)
- void visitVisibilityBlock(VisibilityBlockStmt*)
- void visitBlock(BlockStmt*)
- void visitVariableDecl(VariableDeclStmt*)
- void visitFunctionDef(FunctionDefStmt*)
- void visitReturn(ReturnStmt*)
- void visitExpressionStmt(ExpressionStmt*)
- void visitBranch(BranchStmt*)
- void visitIO(IOStmt*)
- void error(message)
- string typeOf(expr)
- void visit(node)
+ void analyze()
+ bool hasErrors()
+ void printErrors()


abstract
ASTNode
Methods
+ nlohmann::json toJson()


SymbolTable
Attributes
- vector<map<string, SymbolEntry*>> scopes
- int currentLevel
-map<string, vector<SpeciesMemberInfo*>> speciesMembers
- string currentSpeciesContext
Methods
+ void enterScope()
+ void exitScope()
+ bool define(name, type, kind, visibility, parentSpecies)
+ SymbolEntry* lookup(name)
+ int getCurrentLevel()
+ bool hasSpeciesMember(speciesName, memberName)
+ string getSpeciesMemberType(speciesName, memberName)

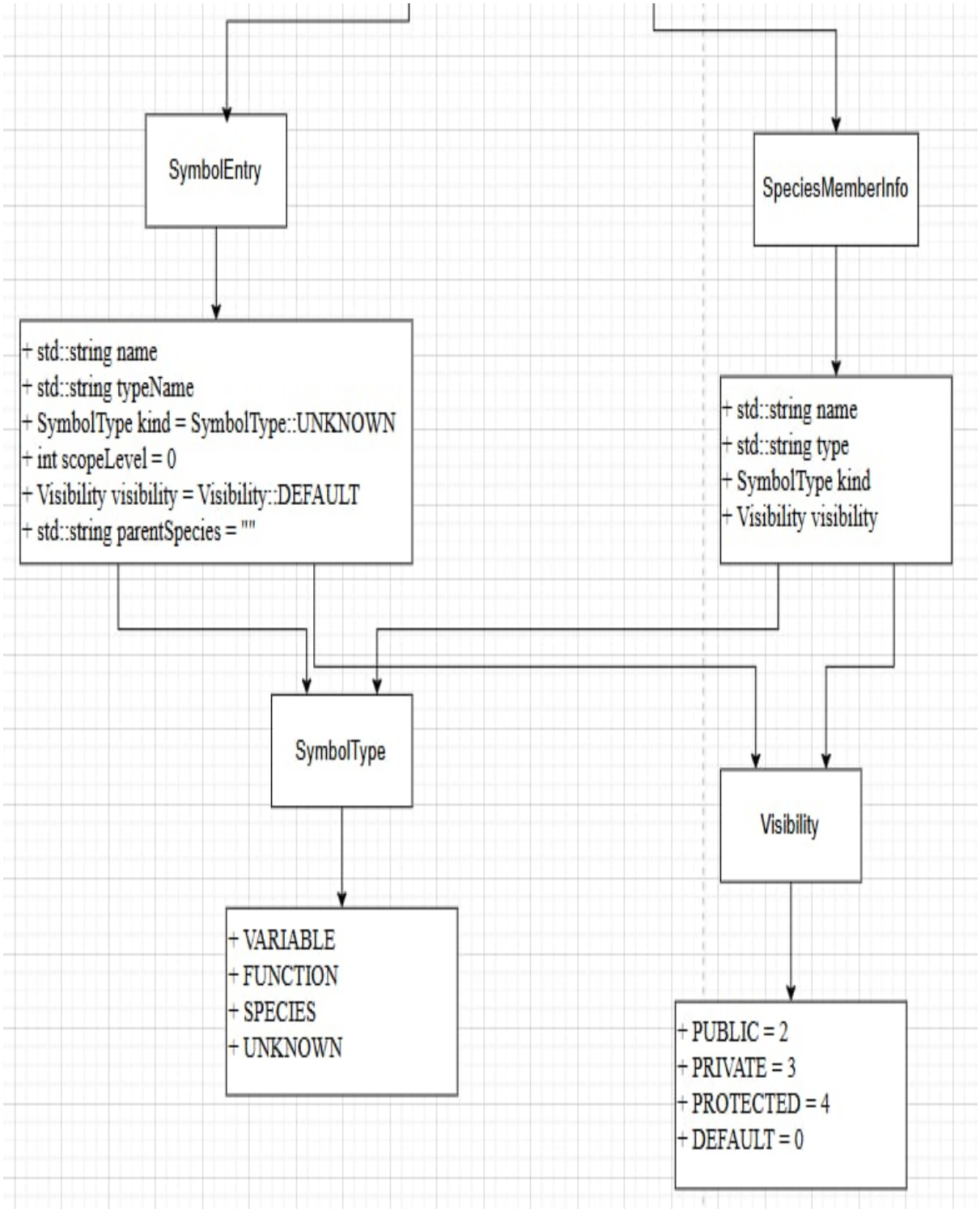


Figure 2: Class diagram of the Semantic Analyzer visitor, symbol table, and related entries.

D Parser Structure (Part 1)

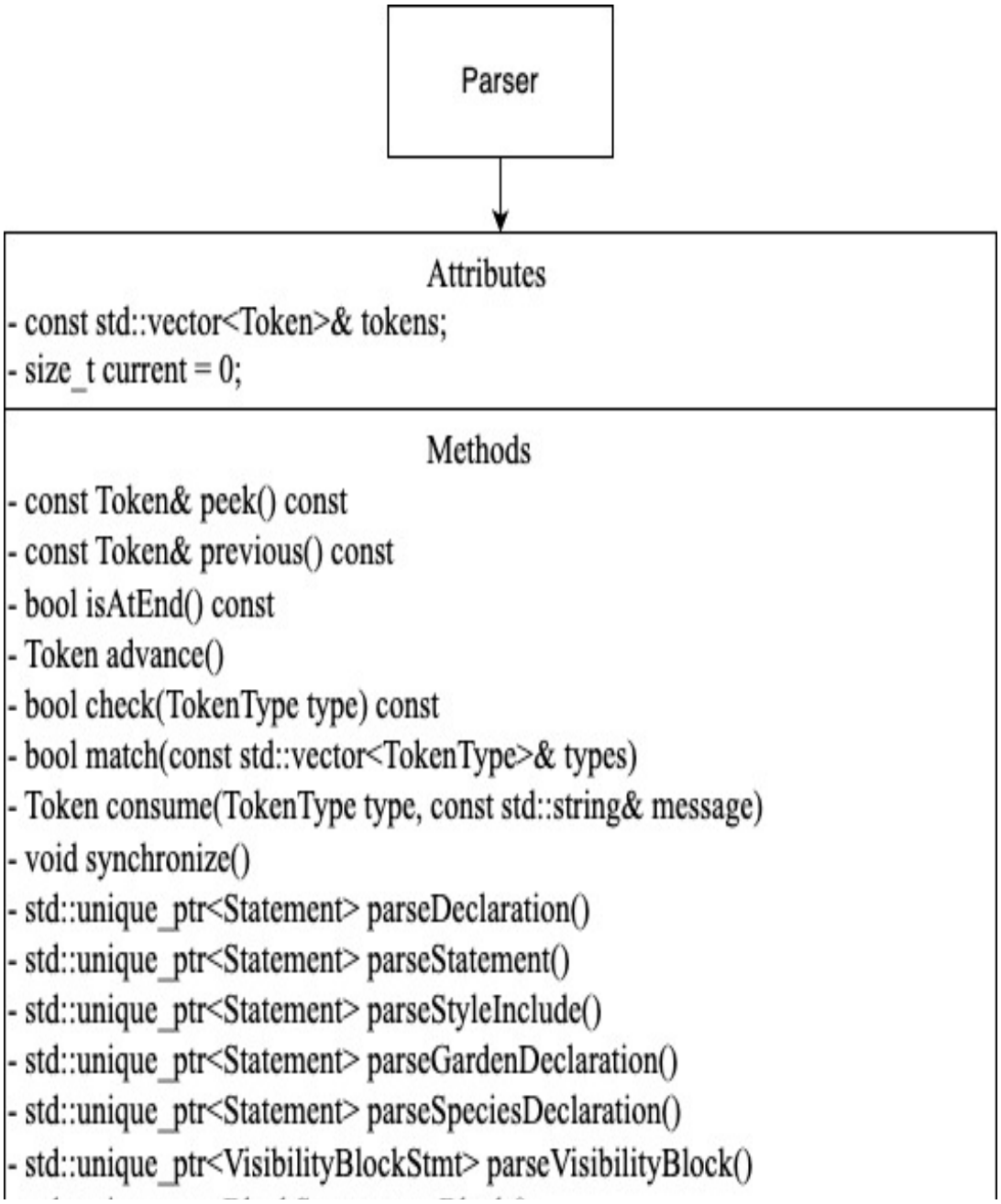


Figure 3: Parser class attributes and parsing methods (overview, part 1).

E Parser Structure (Part 2)

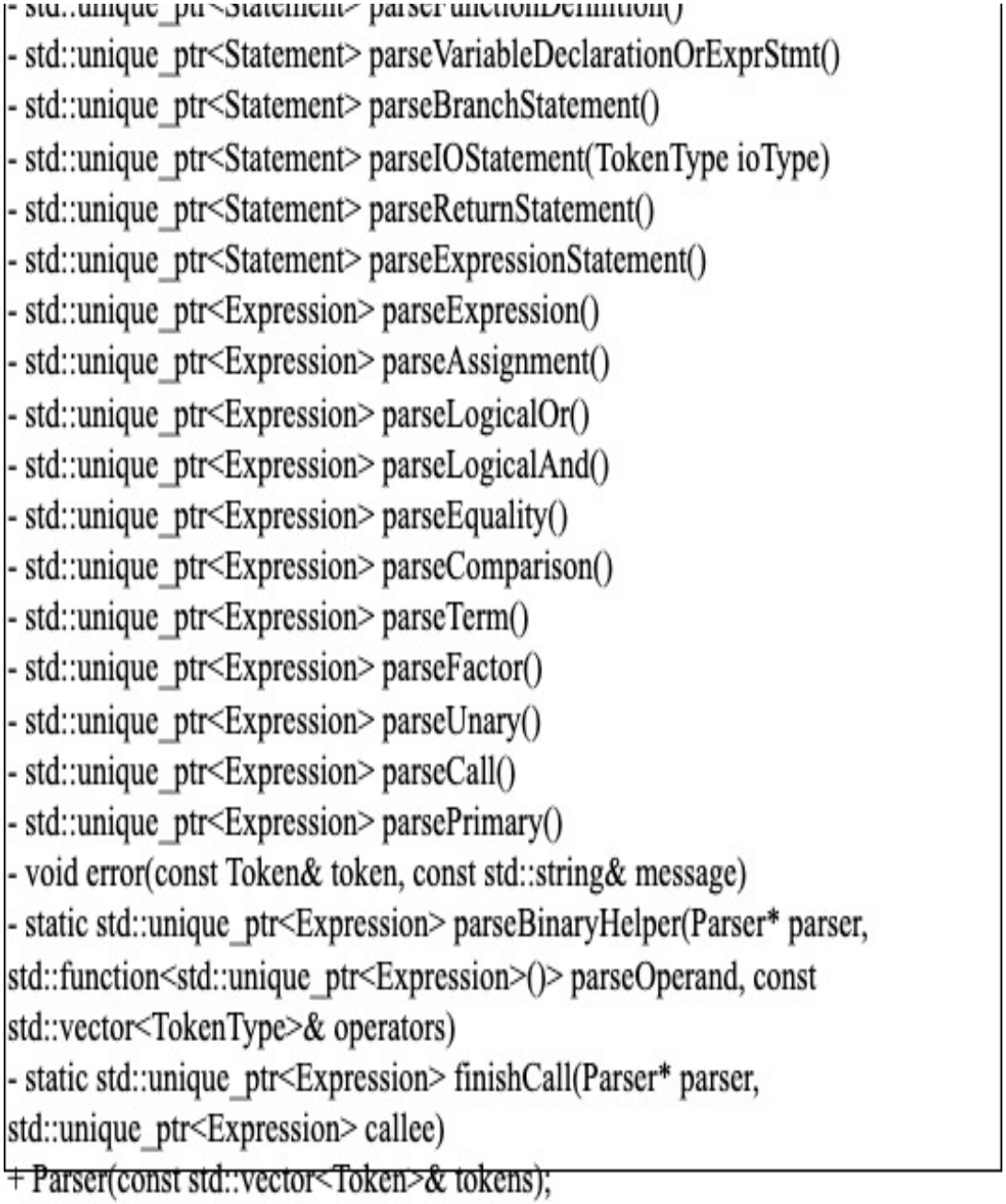
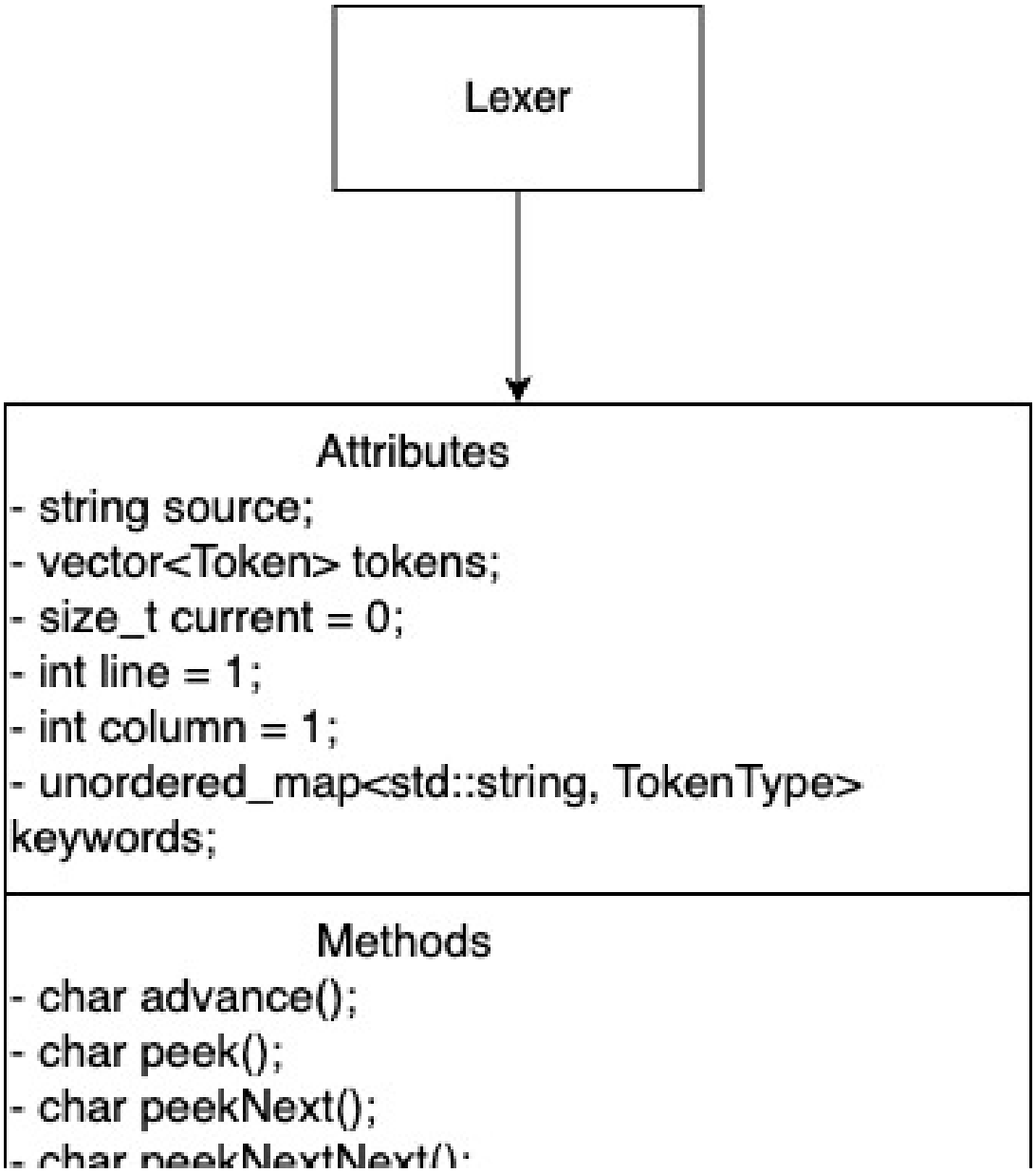


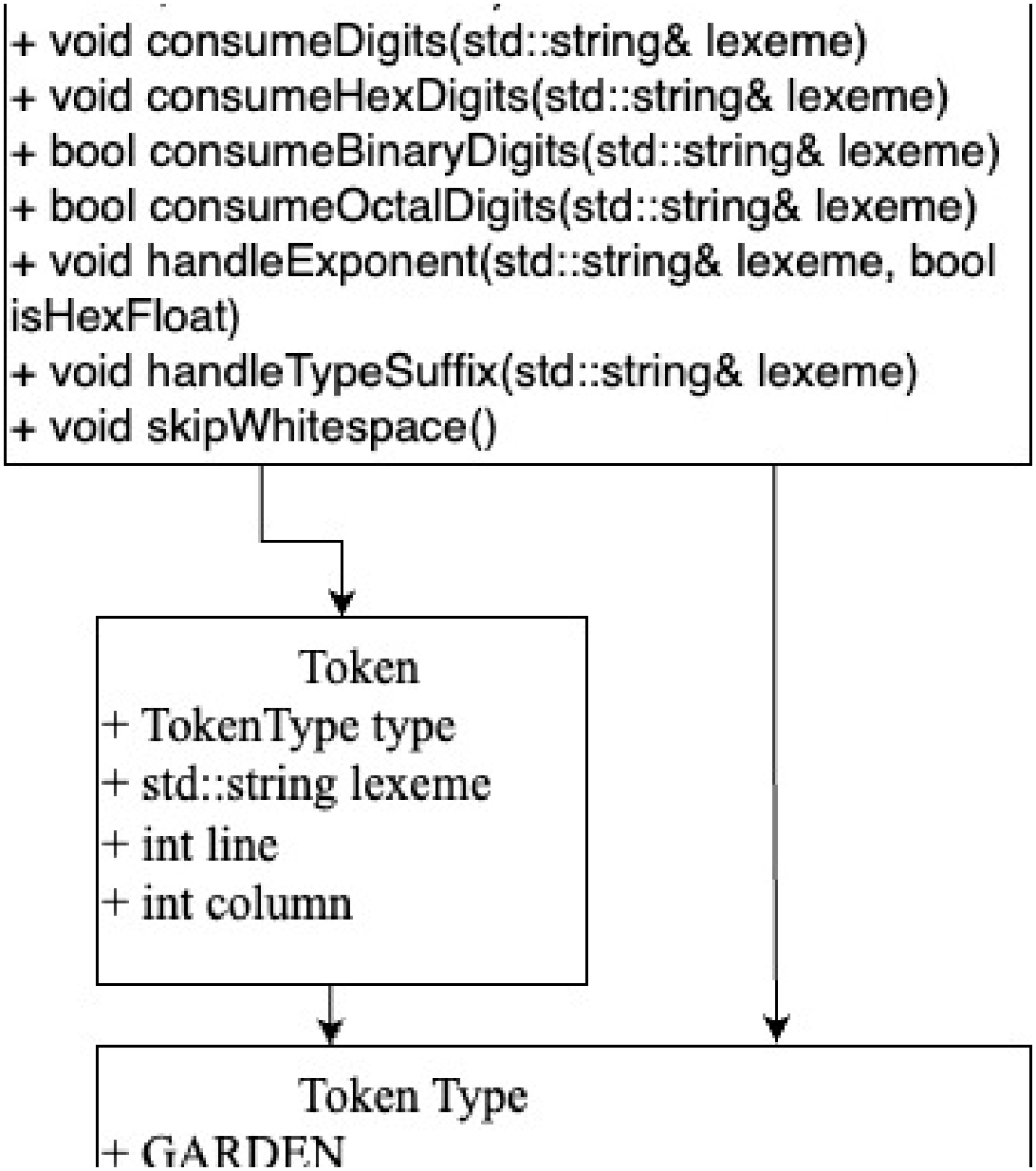
Figure 4: Parser class attributes and parsing methods (details, part 2).

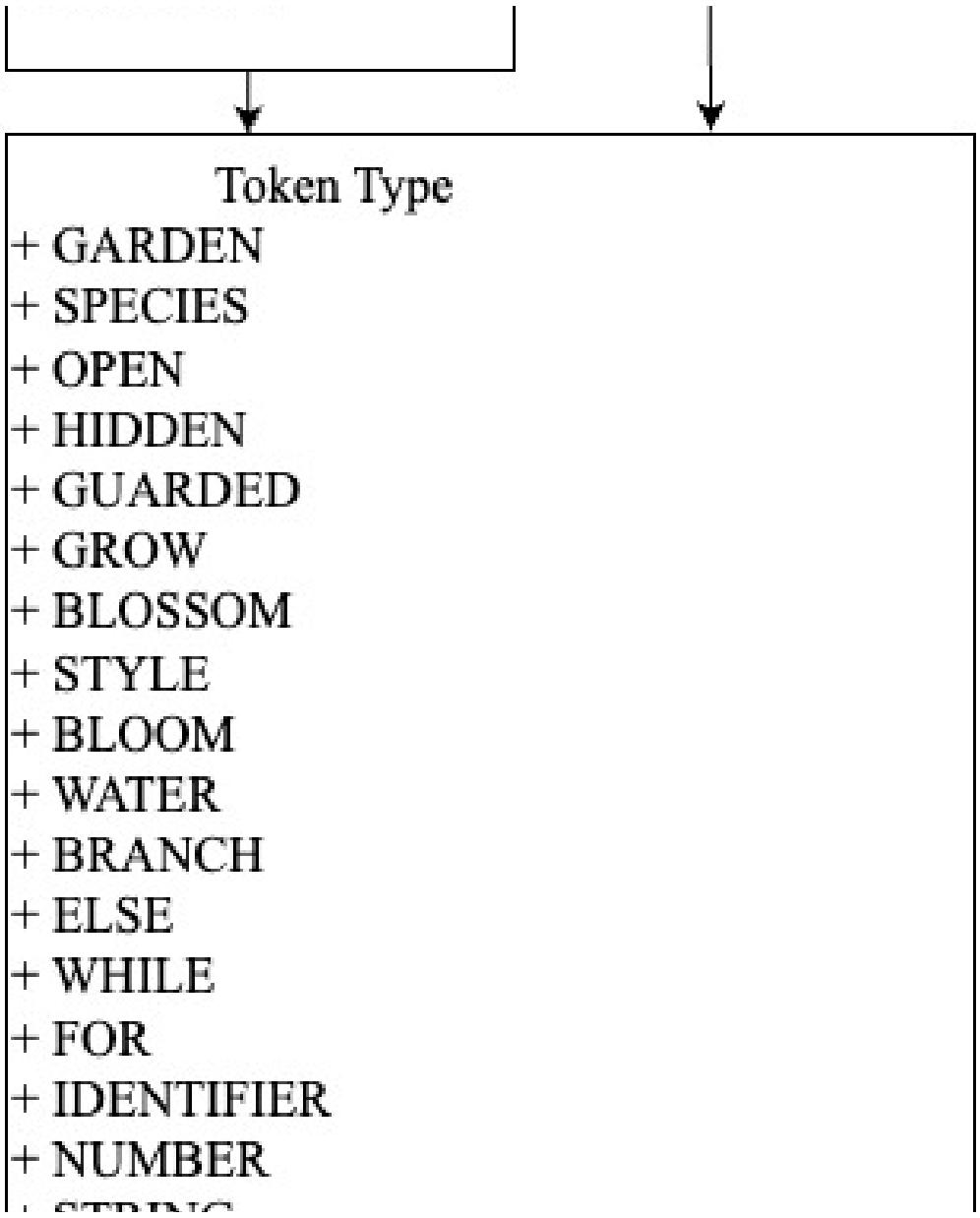
F Lexer Implementation



Methods

- char advance();
- char peek();
- char peekNext();
- char peekNextNext();
- bool isEnd();
- bool match(char expected);
+ Lexer(const std::string& source)
+ std::vector<Token> scanTokens()
+ Token scanToken()
+ void initKeywords()
+ Token identifier()
+ Token Number()
+ Token string()
+ Token skipComment()
+ Token handleSpecialNumber(std::string& lexeme, int startColumn)
+ void consumeDigits(std::string& lexeme)
+ void consumeHexDigits(std::string& lexeme)
+ bool consumeBinaryDigits(std::string& lexeme)
+ bool consumeOctalDigits(std::string& lexeme)


+ void consumeDigits(std::string& lexeme)
+ void consumeHexDigits(std::string& lexeme)
+ bool consumeBinaryDigits(std::string& lexeme)
+ bool consumeOctalDigits(std::string& lexeme)
+ void handleExponent(std::string& lexeme, bool isHexFloat)
+ void handleTypeSuffix(std::string& lexeme)
+ void skipWhitespace()
Token
+ TokenType type
+ std::string lexeme
+ int line
+ int column
Token Type
+ GARDEN


Token Type
+ GARDEN
+ SPECIES
+ OPEN
+ HIDDEN
+ GUARDED
+ GROW
+ BLOSSOM
+ STYLE
+ BLOOM
+ WATER
+ BRANCH
+ ELSE
+ WHILE
+ FOR
+ IDENTIFIER
+ NUMBER

+ WHILE
+ FOR
+ IDENTIFIER
+ NUMBER
+ STRING
+ TRUE
+ FALSE
+ PLUS
+ MINUS
+ STAR
+ SLASH
+ ASSIGN
+ EQUAL
+ NOT_EQUAL
+ LESS
+ LESS_EQUAL
+ GREATER
+ GREATER_EQUAL
+ AND
+ OR

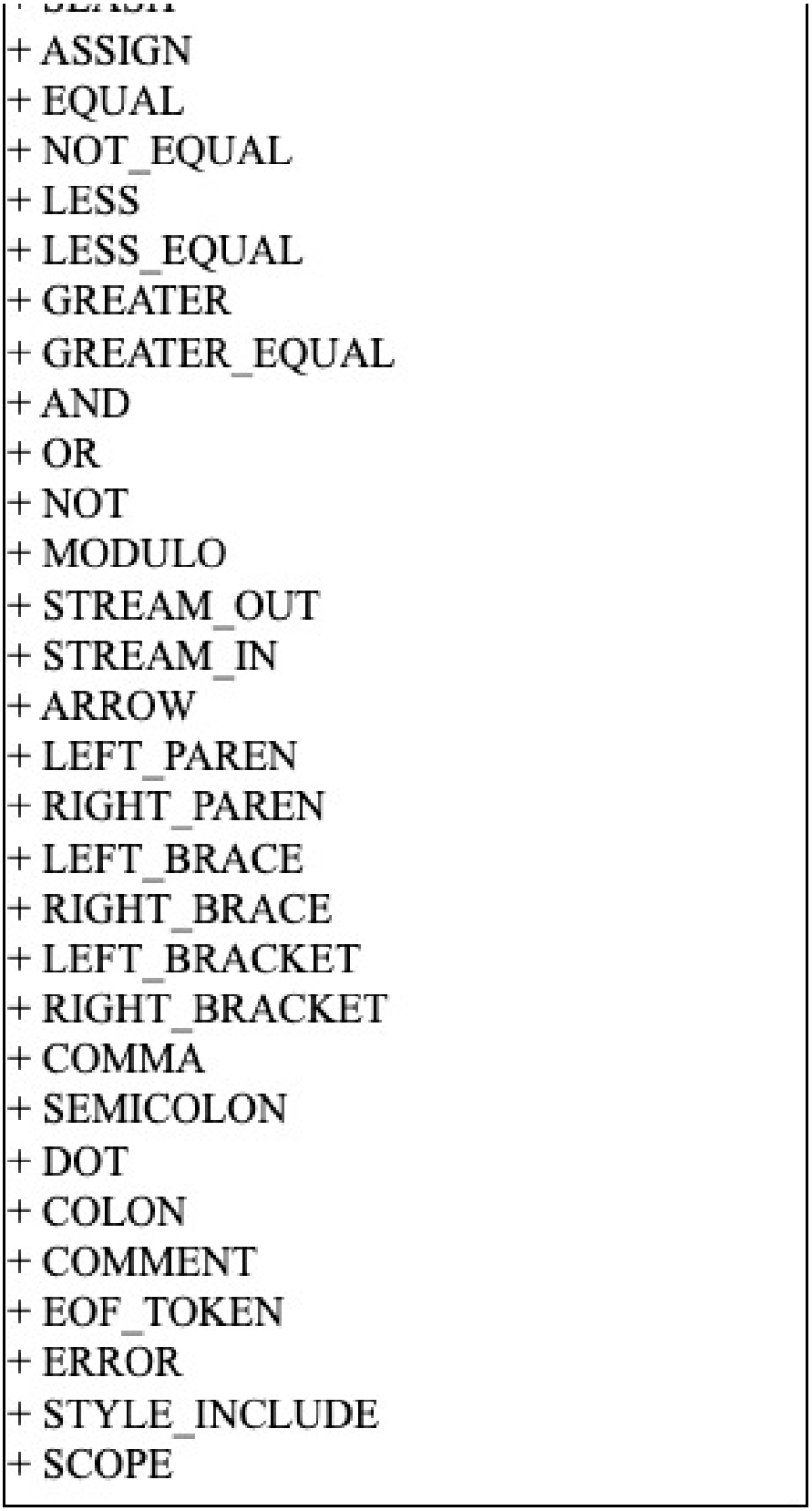
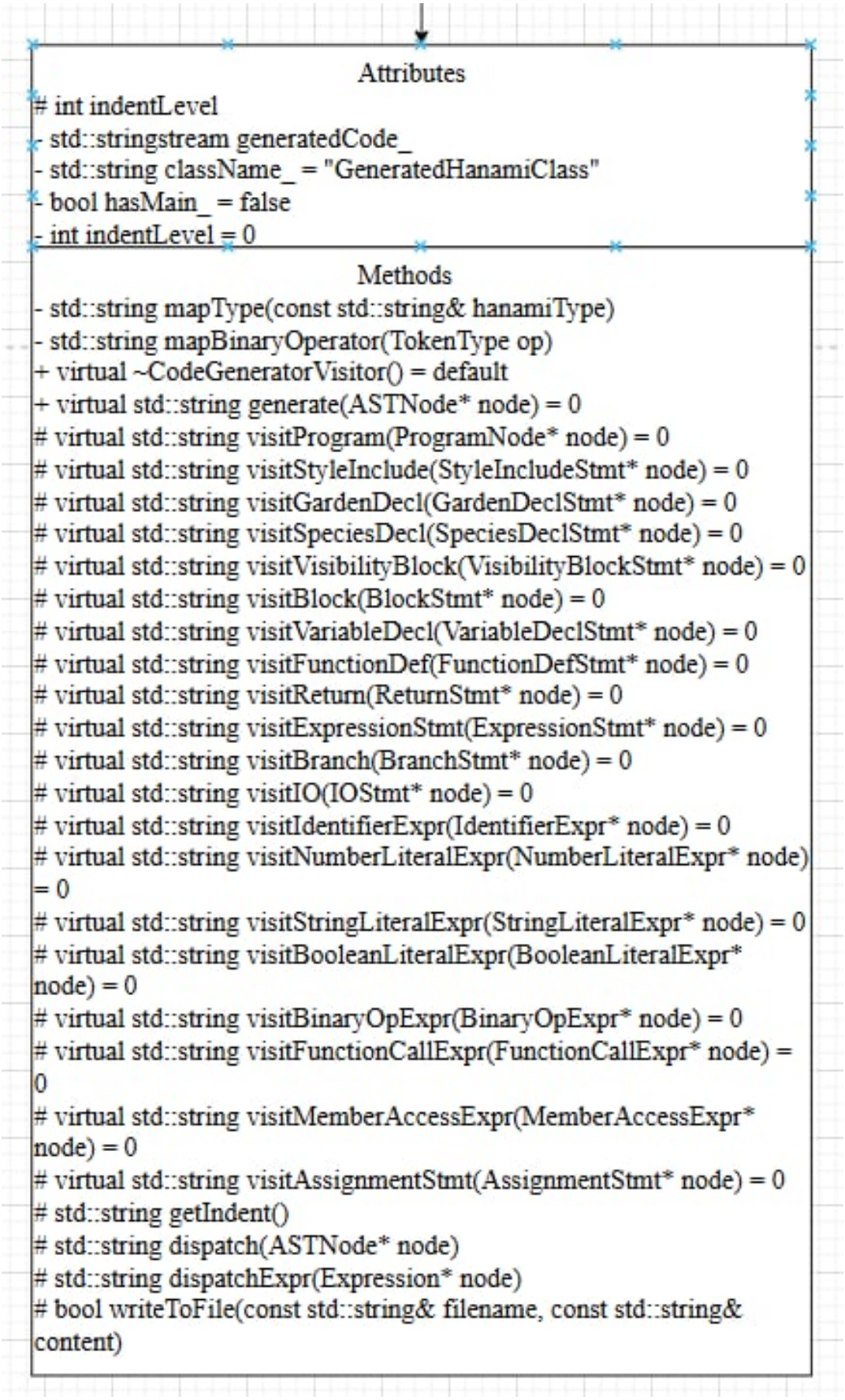


Figure 5: Complete `Lexer` class UML split across six pages to show all attributes, methods, the `Token` struct, and the entire `TokenType` enumeration (from `RIGHT_BRACE` through `SCOPE`).

G Code Generator Visitor


Attributes
int indentLevel
- std::stringstream generatedCode_
- std::string className_ = "GeneratedHanamiClass"
- bool hasMain_ = false
- int indentLevel = 0
Methods
- std::string mapType(const std::string& hanamiType)
- std::string mapBinaryOperator(TokenType op)
+ virtual ~CodeGeneratorVisitor() = default
+ virtual std::string generate(ASTNode* node) = 0
# virtual std::string visitProgram(ProgramNode* node) = 0
# virtual std::string visitStyleInclude(StyleIncludeStmt* node) = 0
# virtual std::string visitGardenDecl(GardenDeclStmt* node) = 0
# virtual std::string visitSpeciesDecl(SpeciesDeclStmt* node) = 0
# virtual std::string visitVisibilityBlock(VisibilityBlockStmt* node) = 0
# virtual std::string visitBlock(BlockStmt* node) = 0
# virtual std::string visitVariableDecl(VariableDeclStmt* node) = 0
# virtual std::string visitFunctionDef(FunctionDefStmt* node) = 0
# virtual std::string visitReturn(ReturnStmt* node) = 0
# virtual std::string visitExpressionStmt(ExpressionStmt* node) = 0
# virtual std::string visitBranch(BranchStmt* node) = 0
# virtual std::string visitIO(IOStmt* node) = 0
# virtual std::string visitIdentifierExpr(IdentifierExpr* node) = 0
# virtual std::string visitNumberLiteralExpr(NumberLiteralExpr* node) = 0
# virtual std::string visitStringLiteralExpr(StringLiteralExpr* node) = 0
# virtual std::string visitBooleanLiteralExpr(BooleanLiteralExpr* node) = 0
# virtual std::string visitBinaryOpExpr(BinaryOpExpr* node) = 0
# virtual std::string visitFunctionCallExpr(FunctionCallExpr* node) = 0
# virtual std::string visitMemberAccessExpr(MemberAccessExpr* node) = 0
# virtual std::string visitAssignmentStmt(AssignmentStmt* node) = 0
# std::string getIndent()
# std::string dispatch(ASTNode* node)
# std::string dispatchExpr(Expression* node)
# bool writeToFile(const std::string& filename, const std::string& content)


Attributes
int indentLevel
- std::stringstream generatedCode_
- std::string currentSpeciesName_
Methods
- std::string mapType(const std::string& hanamiType)
- std::string mapBinaryOperator(TokenType op)
+ virtual ~CodeGeneratorVisitor() = default
+ virtual std::string generate(ASTNode* node) = 0
# virtual std::string visitProgram(ProgramNode* node) = 0
# virtual std::string visitStyleInclude(StyleIncludeStmt* node) = 0
# virtual std::string visitGardenDecl(GardenDeclStmt* node) = 0
# virtual std::string visitSpeciesDecl(SpeciesDeclStmt* node) = 0
# virtual std::string visitVisibilityBlock(VisibilityBlockStmt* node) = 0
# virtual std::string visitBlock(BlockStmt* node) = 0
# virtual std::string visitVariableDecl(VariableDeclStmt* node) = 0
# virtual std::string visitFunctionDef(FunctionDefStmt* node) = 0
# virtual std::string visitReturn(ReturnStmt* node) = 0
# virtual std::string visitExpressionStmt(ExpressionStmt* node) = 0
# virtual std::string visitBranch(BranchStmt* node) = 0
# virtual std::string visitIO(IOStmt* node) = 0
# virtual std::string visitIdentifierExpr(IdentifierExpr* node) = 0
# virtual std::string visitNumberLiteralExpr(NumberLiteralExpr* node) = 0
# virtual std::string visitStringLiteralExpr(StringLiteralExpr* node) = 0
# virtual std::string visitBooleanLiteralExpr(BooleanLiteralExpr* node) = 0
# virtual std::string visitBinaryOpExpr(BinaryOpExpr* node) = 0
# virtual std::string visitFunctionCallExpr(FunctionCallExpr* node) = 0
# virtual std::string visitMemberAccessExpr(MemberAccessExpr* node) = 0
# virtual std::string visitAssignmentStmt(AssignmentStmt* node) = 0
# std::string getIndent()
# std::string dispatch(ASTNode* node)
# std::string dispatchExpr(Expression* node)
# bool writeToFile(const std::string& filename, const std::string& content)

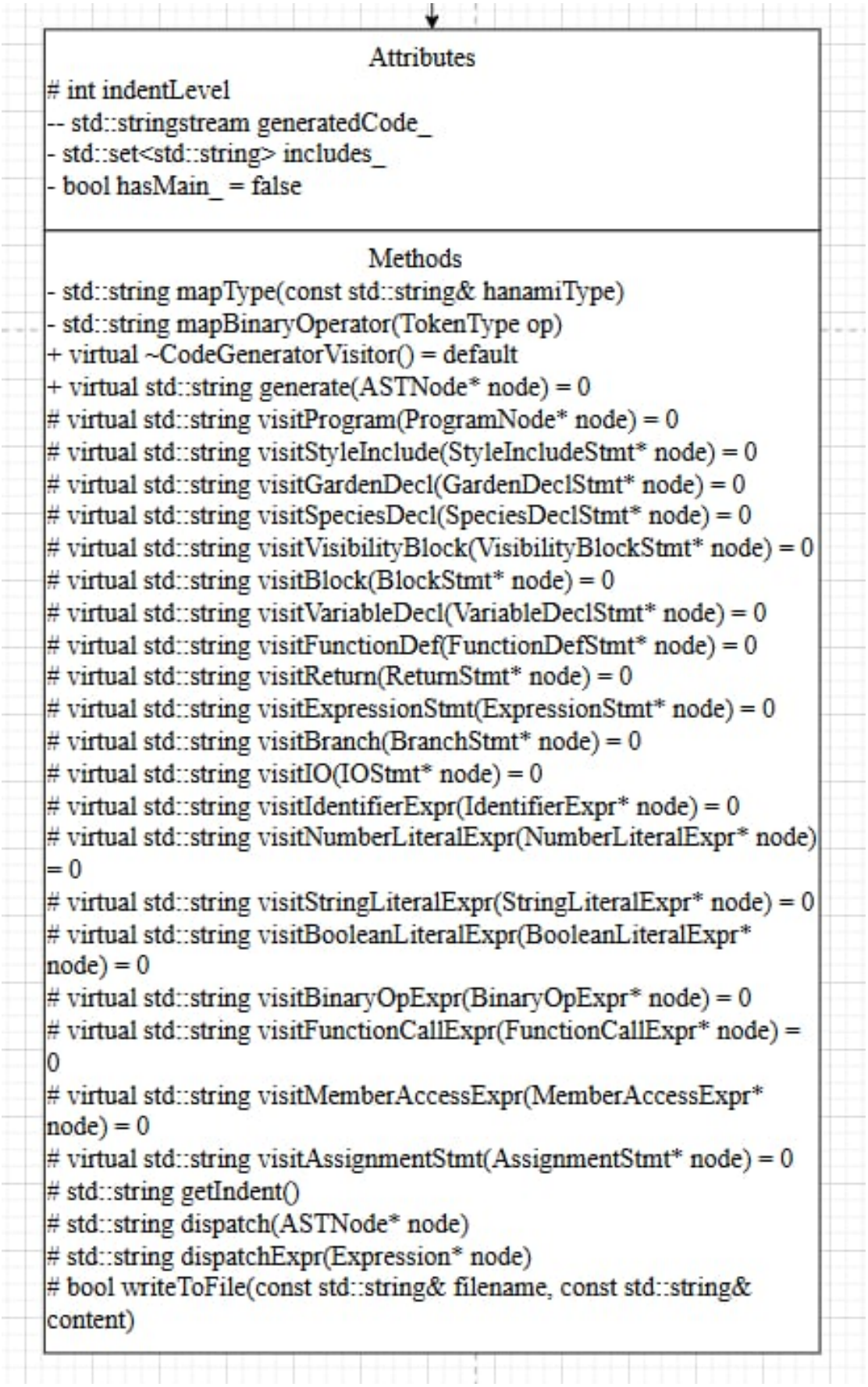

Attributes
int indentLevel
-- std::stringstream generatedCode_
- std::set<std::string> includes_
- bool hasMain_ = false
Methods
- std::string mapType(const std::string& hanamiType)
- std::string mapBinaryOperator(TokenType op)
+ virtual ~CodeGeneratorVisitor() = default
+ virtual std::string generate(ASTNode* node) = 0
# virtual std::string visitProgram(ProgramNode* node) = 0
# virtual std::string visitStyleInclude(StyleIncludeStmt* node) = 0
# virtual std::string visitGardenDecl(GardenDeclStmt* node) = 0
# virtual std::string visitSpeciesDecl(SpeciesDeclStmt* node) = 0
# virtual std::string visitVisibilityBlock(VisibilityBlockStmt* node) = 0
# virtual std::string visitBlock(BlockStmt* node) = 0
# virtual std::string visitVariableDecl(VariableDeclStmt* node) = 0
# virtual std::string visitFunctionDef(FunctionDefStmt* node) = 0
# virtual std::string visitReturn(ReturnStmt* node) = 0
# virtual std::string visitExpressionStmt(ExpressionStmt* node) = 0
# virtual std::string visitBranch(BranchStmt* node) = 0
# virtual std::string visitIO(IOStmt* node) = 0
# virtual std::string visitIdentifierExpr(IdentifierExpr* node) = 0
# virtual std::string visitNumberLiteralExpr(NumberLiteralExpr* node) = 0
# virtual std::string visitStringLiteralExpr(StringLiteralExpr* node) = 0
# virtual std::string visitBooleanLiteralExpr(BooleanLiteralExpr* node) = 0
# virtual std::string visitBinaryOpExpr(BinaryOpExpr* node) = 0
# virtual std::string visitFunctionCallExpr(FunctionCallExpr* node) = 0
# virtual std::string visitMemberAccessExpr(MemberAccessExpr* node) = 0
# virtual std::string visitAssignmentStmt(AssignmentStmt* node) = 0
# std::string getIndent()
# std::string dispatch(ASTNode* node)
# std::string dispatchExpr(Expression* node)
# bool writeToFile(const std::string& filename, const std::string& content)

Figure 6: Abstract `CodeGeneratorVisitor`.

H IR-Based Code Generator

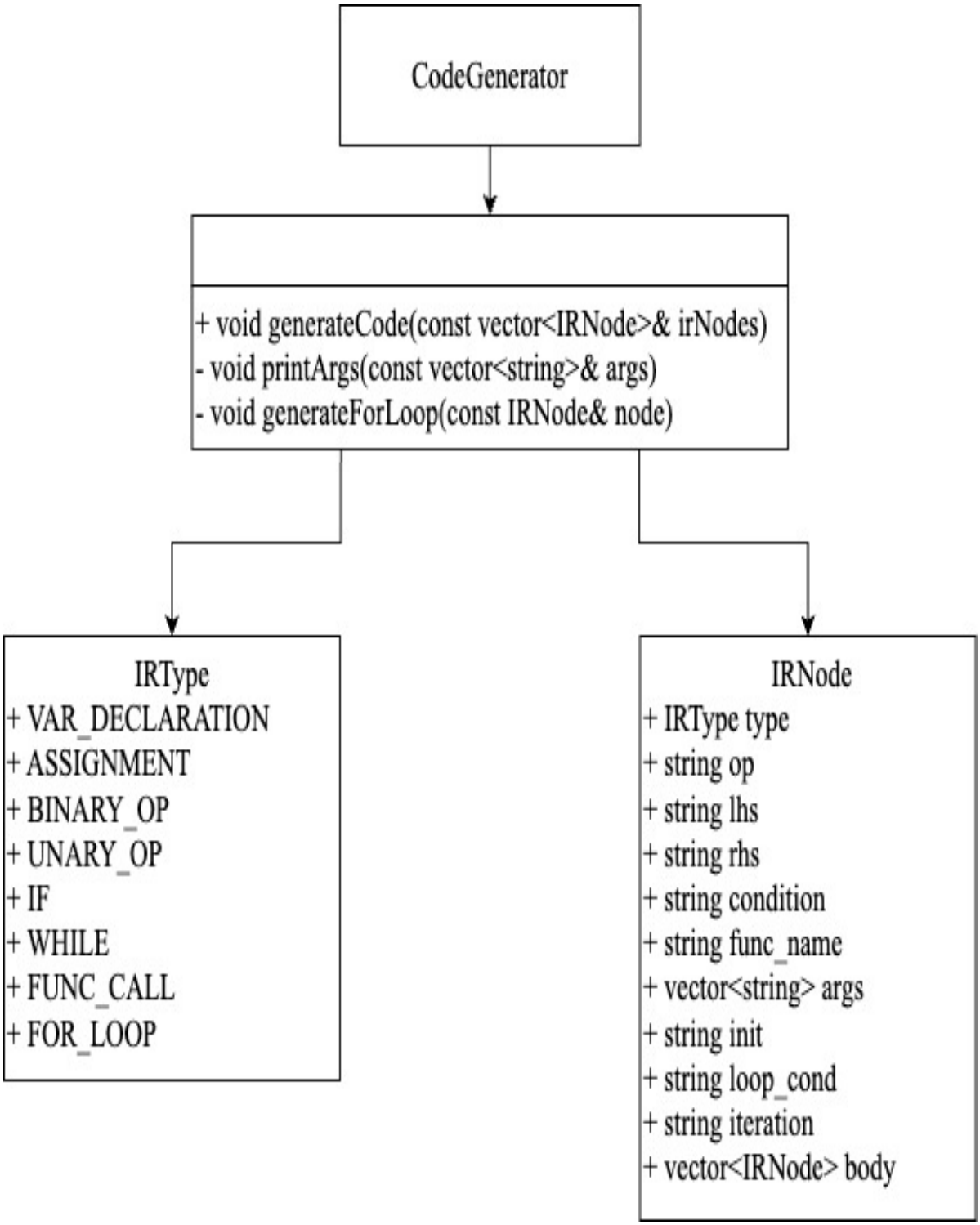


Figure 7: CodeGenerator class overview with IR types and generation routines.

I Sequence Diagram

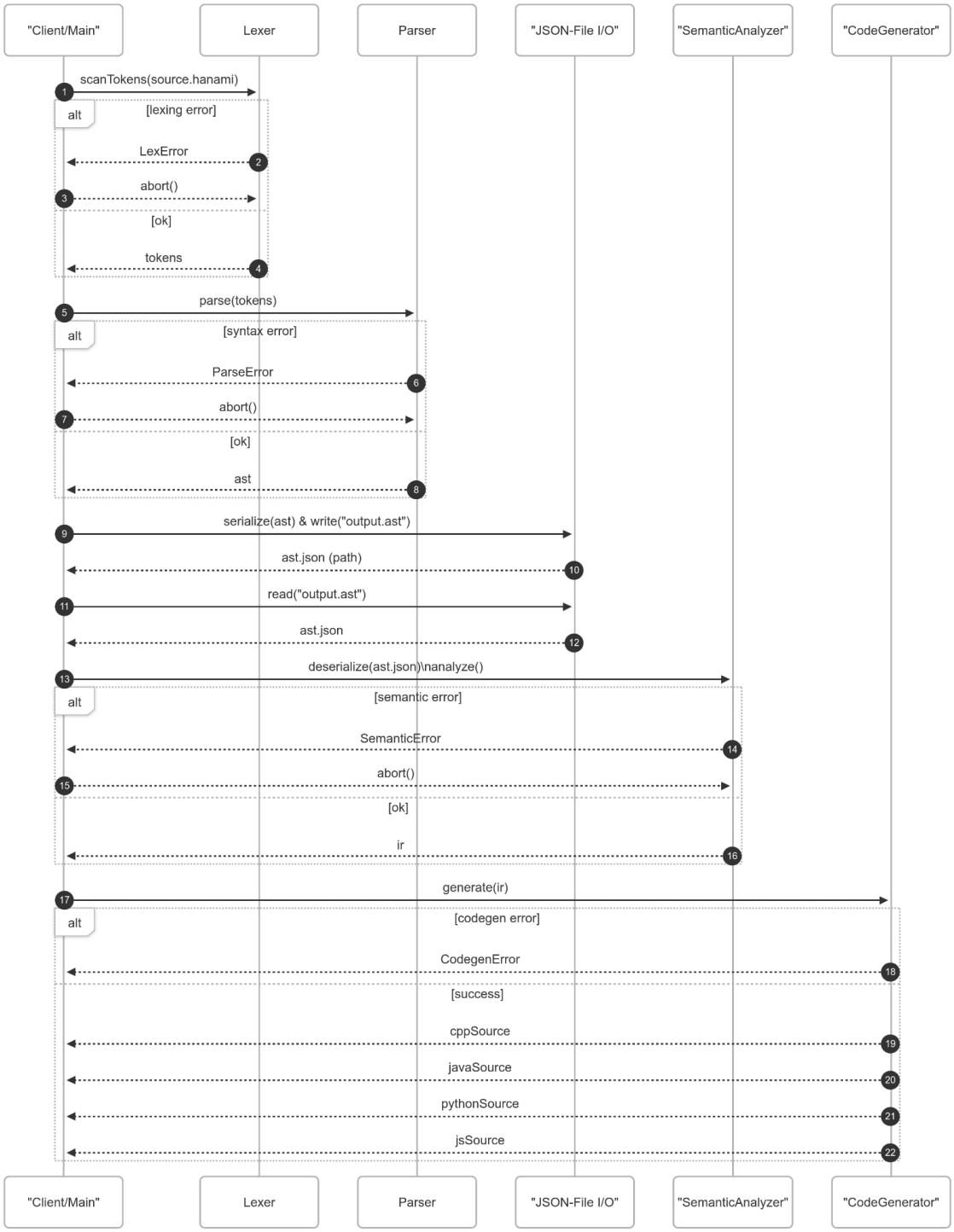


Figure 8: End-to-end sequence diagram depicting interactions between Client/Main, Lexer, Parser, JSON I/O, Semantic Analyzer, and Code Generator.