1Lexical Grammar

1// Whitespace -- IGNORED
2" " | \t | \n | \r | \f
3
4// Single line comment -- IGNORED
5"//" { ~( \n | \r ) } ( \n | \r [\n] )
6
7// Reserved words
8BOOLEAN ::= "boolean"
9ELSE ::= "else"
10FALSE ::= "false"
11IF ::= "if"
12INT ::= "int"
13RETURN ::= "return"
14TRUE ::= "true"
15VOID ::= "void"
16WHILE ::= "while"
17
18// Separators
19COMMA ::= ","
20LCURLY ::= "{"
21LPAREN ::= "("
22RCURLY ::= "}"
23RPAREN ::= ")"
24SEMI ::= ";
25
26// Operators
27ASSIGN ::= "="
28DIV ::= "/"
29EQUAL ::= "=="
30GE ::= ">="
31GT ::= ">
32LAND ::= "&&"
33LE ::= "<="
34LT ::= "<"
35LNOT ::= "!
36LOR ::= "||"
37MINUS ::= "-
38NOT_EQUAL ::= "!="
39PLUS ::= "+
40REM ::= "%"
41STAR ::= "*
42
43// Identifiers
44IDENTIFIER ::= ( "a"..."z" | "A"..."Z" | "_" | "$" )
45{ "a"..."z" | "A"..."Z" | "_" | "0"..."9" | "$" }
46
47// Literals
48BOOLEAN_LITERAL ::= "false" | "true"
49INT_LITERAL ::= ( "0"..."9" ) { "0"..."9" }
50
51// End of file
52EOF ::= "<end of file>"
2 Syntactic Grammar

```
compilationUnit ::= { methodDeclaration } EOF

methodDeclaration ::= ( VOID | type ) IDENTIFIER formalParameters block

formalParameters ::= LPAREN [ formalParameter { COMMA formalParameter } ] RPAREN

formalParameter ::= type IDENTIFIER

block ::= LCURLY { statement } RCURLY

statement ::= block

| type IDENTIFIER [ ASSIGN expression ] SEMI
| IF parExpression statement [ ELSE statement ]
| RETURN [ expression ] SEMI
| WHILE parExpression statement
| statementExpression SEMI

parExpression ::= LPAREN expression RPAREN

type ::= BOOLEAN | INT

statementExpression ::= expression

expression ::= assignmentExpression

assignmentExpression ::= conditionalOrExpression [ ASSIGN assignmentExpression ]

conditionalOrExpression ::= conditionalAndExpression { LOR conditionalAndExpression }

conditionalAndExpression ::= equalityExpression { LAND equalityExpression }

equalityExpression ::= relationalExpression { ( EQUAL | NOT_EQUAL ) relationalExpression }

relationalExpression ::= additiveExpression { ( GE | GT | LE | LT ) additiveExpression }

additiveExpression ::= multiplicativeExpression { ( MINUS | PLUS ) multiplicativeExpression }

multiplicativeExpression ::= unaryExpression { ( DIV | REM | STAR ) unaryExpression }

unaryExpression ::= ( LNOT | MINUS ) unaryExpression
| parExpression
| IDENTIFIER [ arguments ]
| literal

arguments ::= LPAREN [ expression { COMMA expression } ] RPAREN

literal ::= BOOLEAN_LITERAL | INT_LITERAL
```

The method `void main() { ... }` serves as the entry-point method, i.e., is invoked when the program is run. If this method is not explicitly defined in the compilation unit, an implicit entry-point method with an empty body is provided.
3 Semantics

```plaintext
IBinaryExpression:
  - lhs and rhs must be integers
  - IAssignOp:
    - lhs must be a variable
    - lhs and rhs must have the same type

IBooleanBinaryExpression:
  - IEqualOp, INotEqualOp:
    - lhs and rhs must have the same type
  - ILogicalAndOp, ILogicalOrOp:
    - lhs and rhs must be booleans

IComparisonExpression:
  - lhs and rhs must be integers

IMessageExpression:
  - The message must correspond to a valid method

IMethodDeclaration:
  - Must not be defined already
  - Non-void method must have a return statement

IIfStatement:
  - The condition must be a boolean

IUnaryExpression:
  - ILogicalNotOp
    - The operand must be a boolean
  - INegateOp
    - The operand must be an integer

IVariable:
  - The variable must be declared
  - The variable must be initialized

IVariableDeclaration:
  - The variable must not shadow another local variable

IReturnStatement:
  - Must not return a value from a void method
  - The type of return value in a non-void method must match return type of the method
  - A non-void method must return a value

IWhileStatement:
  - The condition must be a boolean
```

4 Input and Output

The *iota* language supports the following builtin methods for reading from standard input and writing to standard output:
// Reads and returns an int from standard input.
int read()

// Writes x (1 for true and 0 for false) to standard output.
void write(boolean x)

// Writes x to standard output.
void write(int x)