

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

2 Syntactic Grammar

```

1 compilationUnit ::= { methodDeclaration } EOF
2
3 methodDeclaration ::= ( VOID | type ) IDENTIFIER formalParameters block
4
5 formalParameters ::= LPAREN [ formalParameter { COMMA formalParameter } ] RPAREN
6
7 formalParameter ::= type IDENTIFIER
8
9 block ::= LCURLY { statement } RCURLY
10
11 statement ::= block
12             | type IDENTIFIER [ ASSIGN expression ] SEMI
13             | IF parExpression statement [ ELSE statement ]
14             | RETURN [ expression ] SEMI
15             | WHILE parExpression statement
16             | statementExpression SEMI
17
18 parExpression ::= LPAREN expression RPAREN
19
20 type ::= BOOLEAN | INT
21
22 statementExpression ::= expression
23
24 expression ::= assignmentExpression
25
26 assignmentExpression ::= conditionalOrExpression [ ASSIGN assignmentExpression ]
27
28 conditionalOrExpression ::= conditionalAndExpression { LOR conditionalAndExpression }
29
30 conditionalAndExpression ::= equalityExpression { LAND equalityExpression }
31
32 equalityExpression ::= relationalExpression { ( EQUAL | NOT_EQUAL ) relationalExpression }
33
34 relationalExpression ::= additiveExpression [ ( GE | GT | LE | LT ) additiveExpression ]
35
36 additiveExpression ::= multiplicativeExpression { ( MINUS | PLUS ) multiplicativeExpression }
37
38 multiplicativeExpression ::= unaryExpression { ( DIV | REM | STAR ) unaryExpression }
39
40 unaryExpression ::= ( LNOT | MINUS ) unaryExpression
41                   | parExpression
42                   | IDENTIFIER [ arguments ]
43                   | literal
44
45 arguments ::= LPAREN [ expression { COMMA expression } ] RPAREN
46
47 literal ::= BOOLEAN_LITERAL | INT_LITERAL

```

The method `void main() { ... }` serves as the entry-point method, ie, 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

```
1  IBinaryExpression:
2  - IDivideOp, IMultiplyOf, IPlusOp, IRemainderOp, ISubtractOp
3    - lhs and rhs must be integers
4  - IAssignOp:
5    - lhs must be a variable
6    - lhs and rhs must have the same type
7
8  IBooleanBinaryExpression:
9  - IEqualOp, INotEqualOp:
10   - lhs and rhs must have the same type
11 - ILogicalAndOp, ILogicalOrOp:
12   - lhs and rhs must be booleans
13
14 IComparisonExpression:
15 - lhs and rhs must be integers
16
17 IMessageExpression:
18 - The message must correspond to a valid method
19
20 IMethodDeclaration:
21 - Must not be defined already
22 - Non-void method must have a return statement
23
24 IIfStatement:
25 - The condition must be a boolean
26
27 IUnaryExpression:
28 - ILogicalNotOp
29   - The operand must be a boolean
30 - INegateOp
31   - The operand must be an integer
32
33 IVariable:
34 - The variable ust be declared
35 - The variable must be initialized
36
37 IVariableDeclaration:
38 - The variable must not shadow another local variable
39
40 IReturnStatement:
41 - Must not return a value from a void method
42 - The type of return value in a non-void method must match return type of the method
43 - A non-void method must return a value
44
45 IWhileStatement:
46 - 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:

```
1 // Reads and returns an int from standard input.  
2 int read()  
3  
4 // Writes x (1 for true and 0 for false) to standard output.  
5 void write(boolean x)  
6  
7 // Writes x to standard output.  
8 void write(int x)
```