ℌℒℒ Language Specification

Draft

Michael Heilmann
michaelheilmann@primordialmachine.com

July 19, 2018

Abstract

Contents

1 Operators

Associativity in ℌℒℒ. Except for the assignment operator and the exponentiation operator, all binary operators are left-to-right-associative. The assignment operator as well as the exponentiation operator are both right-to-left-associative. However, as transitive assignments ($\mathtt{x}_1 := \mathtt{x}_2 := \mathtt{x}_3 := \ldots$) are not allowed in ℌℒℒ, the associativity of the assignment operators is irrelevant.

Precendence in ℌℒℒ. Operators of the same precedence are are in the same precedence group of that precedence.

1.1 Expression Operators

The relative precedence of expression operator precendence groups and the operators enclosed in each group are provided in table **??**. Note that the assignment operator is *not* an expression operator.

Group	**Operator**
Parentheses	`(`*a*`)`
Access Operator	*a*`.`*b*
Exponentiation	*a* `**` *b*
Multiplication	*a* `*` *b* *a* `/` *b* *a* `mod` *b* *a* `rem` *b*
Additive Identity and Negation	`+` *a* `-` *a*
Addition	*a* `+` *b* *a* `-` *b*
Relational Operators	*a* `<` *b* *a* `<=` *b* *a* `>` *b* *a* `>=` *b*
Equality Operators	*a* `/=` *b* *a* `=` *b*
Logical Conjunction	`and` *a*
Logical Disjunction	`or` *a*
Logical Negation	`not` *a*

Table 1: Expression Operator Precedence Groups.

1.1.1 Statement operators

2 Lexical Grammar

2.1 Names and Literals

$$
\begin{aligned}
\text{digit} &\rightarrow \mathbf{0} - \mathbf{9} \\
\text{alphatic} &\rightarrow \mathbf{a} - \mathbf{z} \mid \mathbf{A} - \mathbf{Z} \\
\text{exponent} &\rightarrow (\mathbf{E} \mid \mathbf{e})\ (\texttt{+} \mid \texttt{-})^{?}\ \text{digit}^{+} \\
\text{integer} &\rightarrow \text{digit}^{+}\text{exp}^{?} \\
\text{real} &\rightarrow \text{real}_1 \mid \text{real}_2 \\
\text{real}_1 &\rightarrow \text{digit}^{*}\ \mathbf{.}\ \text{digit}^{+}\ \text{exp}^{?} \\
\text{real}_2 &\rightarrow \text{digit}^{+}\ \mathbf{.}\ \text{digit}^{*}\ \text{exp}^{?} \\
\text{name} &\rightarrow \texttt{\_}^{*}\ \text{alpha}\ (\text{alpha} \mid \text{digit} \mid \texttt{\_})^{*} \\
\text{boolean} &\rightarrow \mathbf{true} \mid \mathbf{false}
\end{aligned}
$$

2.2 Operators

Note that - does not follow the usual naming schema of lexical items classified as operators as it may be interpreted as arithmetic negation or subtraction in later stages.

$$
\begin{aligned}
\text{add} &\rightarrow \texttt{+} \\
\text{minus} &\rightarrow \texttt{-} \\
\text{multiply} &\rightarrow \texttt{*} \\
\text{divide} &\rightarrow \texttt{/} \\
\text{access} &\rightarrow \mathbf{.} \\
\text{assign} &\rightarrow \texttt{:=} \\
\text{equal} &\rightarrow \texttt{=} \\
\text{notEqual} &\rightarrow \texttt{/=} \\
\text{lowerThan} &\rightarrow \texttt{<} \\
\text{lowerThanOrEqualTo} &\rightarrow \texttt{<=} \\
\text{greaterThan} &\rightarrow \texttt{>} \\
\text{greaterThanOrEqualTo} &\rightarrow \texttt{>=} \\
\text{and} &\rightarrow \mathbf{and} \\
\text{or} &\rightarrow \mathbf{or} \\
\text{not} &\rightarrow \mathbf{not} \\
\text{exponentiate} &\rightarrow \texttt{**} \\
\text{modulus} &\rightarrow \mathbf{mod} \\
\text{remainder} &\rightarrow \mathbf{rem}
\end{aligned}
$$

2.3 Separators and Delimiters

semicolon	→	**;**
comma	→	**,**
leftParenthesis	→	**(**
rightParenthesis	→	**)**

2.4 Keywords

class	→	**class**
constructor	→	**constructor**
destructor	→	**destructor**
enumeration	→	**enumeration**
interface	→	**interface**
method	→	**method**

3 Syntactical Grammar

3.1 Operators

unaryAdditiveOperator → **minus**
add

equalityOperator → **equal**
| **notEqual**

relationalOperator → **lowerThan**
| **lowerThanOrEqualTo**
| **greaterThan**
| **greaterThanOrEqualTo**

additiveOperator → **add**
| **minus**

multiplicativeOperator → **mulitply**
| **divide**
| **modulus**
| **remainder**

notOperator → **not**

andOperator → **and**

andOperator → **or**

booleanLiteral → **boolean**

characterLiteral → **character**

integerLiteral → **integer**

realLiteral → **real**

stringLiteral → **string**

literal → booleanLiteral
| characterLiteral
| integerLiteral
| realLiteral
| stringLiteral

3.2 Expressions

expression	→	unaryLogicalExpression
unaryLogicalExpression	→	notOperator* logicalExpression
logicalExpression	→	orExpression
orExpression	→	andExpression (orOperator andExpression) *
andExpression	→	equalityExpression (andOperator equalityExpression) *
equalityExpression	→	relationalExpression (equalityOperator relationalExpression) *
relationalExpression	→	additiveExpression (relationalOperator additiveExpression) *
additiveExpression	→	multiplicativeExpression (additiveOperator multiplicativeExpression) *
multiplicativeExpression	→	unaryAdditiveExpression (multiplicativeOperator unaryAdditiveExpression) *
unaryAdditiveExpression	→	unaryAdditiveOperator* exponentiationExpression
exponentiationExpression	→	accessExpression (exponentiationOperator exponentiationExpression) ?
callExpression	→	accessExpression (argumentList) *
accessExpression	→	primaryExpression (accessOperator primaryExpression) *
primaryExpression	→	literal | **name** | **leftParenthesis** expression **rightParenthesis**

argumentList	→	**leftParenthesis** expressionList? **rightParenthesis**
expressionList	→	expression (**comma** expression) *

3.3 Package-Level Elements

packageDeclaration	→	**package** name packageBodyDeclaration
classOrEnumerationOrInterfaceDeclaration	→	classDeclaration
	→	| enumerationDeclaration | interfaceDeclaration
classDeclaration	→	**class** name (**extends** type) ? (**implements** typeList) ? classBodyDeclaration
interfaceDeclaration	→	**interface** name (**extends** typeList) ? interfaceBodyDeclaration

packageBodyDeclaration	→	**is** packageBodyElementDeclaration* **end**
classBodyDeclaration	→	**is** classBodyElementDeclaration* **end**
enumerationBodyDeclaration	→	**is** enumerationBodyElementDeclaration* **end**
interfaceBodyDeclaration	→	**is** interfaceBodyElementDeclaration* **end**

packageBodyElementDeclaration → classDeclaration packageBodyElementDeclaration* **end**
classBodyElementDeclaration → classOrInterfaceOrEnumerationDeclaration
| attributeDeclaration
| constructorDeclaration
| destructorDeclaration
| methodDeclarationInClass
enumerationBodyElementDeclaration → enumerationElementDeclaration
enumerationElementDeclaration → name **assign** expression
interfaceBodyElementDeclaration → classOrInterfaceOrEnumerationDeclaration
| methodDeclarationInInterface

typeList → type (**comma** type)*
type → arrayType
| namedType
arrayType → ([ ])+ namedType
namedType → qualifiedName

3.4 Operator Declarations and, in particular, Constructor and Destructor Declarations

constructorDeclaration → **constructor** parameterListDeclaration
callableBodyDeclaration
destructorDeclaration → **destructor** parameterListDeclaration
callableBodyDeclaration
constructorBodyDeclaration → **is** statementList **end**
destructorBodyDeclaration → **is** statementList **end**

3.5 Method Declarations, Constructor Declarations and Destructor Declarations

methodDeclaration → **method** methodModifiers qualifiedName
parameterListDeclaration type
callableBodyDeclaration
parameterListDeclaration → **leftParenthesis**
(parameterDeclaration (**comma** parameterDeclaration)*)?
rightParenthesis
parameterDeclaration → name type
methodBodyDeclaration → **is** statementList **end**

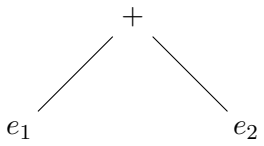

Figure 1: AST-prototype for `add`.

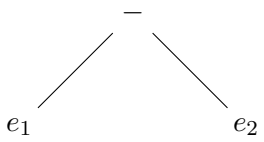

Figure 2: AST-prototype for `sub`.

3.6 Blocks and Statements

block	$\rightarrow$	**is** statementList **end**
statementList	$\rightarrow$	statement*
statement	$\rightarrow$	expression