

A screenshot of a video game

Description automatically generated with medium confidence

# **Julius Language LANGUAGE SPECIFICATION[[1]](#footnote-0)**

### **General View**

This document focuses on **Phonon** LS (Language Specification).

***Grammar, which knows how to control even kings . . .***

*—*Molière*, Les Femmes Savantes* (1672), Act II, scene vi

A context-free grammar is used to define the lexical and syntactical parts of the **Phonon LANGUAGE** and the lexical and syntactic structure of a **Phonon LANGUAGE** program.

1. **The Phonon LANGUAGE Lexical Specification**
   1. **White Space**

White spaceis defined as the ASCII space, horizontal and vertical tabs, and form feed characters, as well as line terminators. White space is discarded by the scanner.

**<white space>** → *one of* { SPACE, TAB, FF, NL, CR, NLCR }

* 1. **Comments**

**Phonon** LANGUAGE supports only single-line comments: all the text from the ASCII characters **!!** to the end of the line is ignored by the scanner.

**<comments>** → ^ { sequence of ASCII chars } ^

* 1. **Variable Identifiers**

The following variable identifier (VID) tokens are produced by the scanner: two kinds of arithmetic tokens: **IVID\_T** (integer) and **FVID\_T** (float-point numbers) and one kind of strings: **SVID\_T**.

**<variable identifier>** → VID\_T

* 1. **Keywords**

The scanner produces a single token: **KW\_T**. The type of the keyword is defined by the attribute of the token (the index of the keywordTable[]). Remember that the list of keywords in **Phonon** language is given by:

**while, if, else, return, void, out, in, float, int, bool**

* 1. **Integer Literals**

The scanner produces a single token: **IL\_T** with an integer value as an attribute.

**<integer\_literal>** → IL\_T

* 1. **Floating-point Literals**

**RL\_T** token with a real decimal value as an attribute is produced by the scanner.

**<float\_literal>** → FL\_T

* 1. **String Literals**

**SL\_T** token is produced by the scanner.

**<string\_literal>** → SL\_T

* 1. **Separators**

**<separator>** → *one of* {(, ), {, }, ,, ; }

Some different tokens are produced by the scanner - **LPR\_T**, **RPR\_T**, **LBR\_T**, **RBR\_T**, **COM\_T**, **EOS\_T**.

* 1. **Operators**

**<arithmetic operator>** → *one of* { +, -, \*, / }

Some different tokens are produced by the scanner - **ADD\_T, SUB\_T, MUL\_T, DIV\_T**

**<string concatenation operator>** → *++*

A single token is produced by the scanner: **LOG\_OP\_T**. The type of the operator is defined by the attribute of the token.

**<assignment operator>** → =

A single token is produced by the scanner: **ASS\_OP\_T**.

1. **The Phonon LANGUAGE Syntactic Specification**
   1. **Phonon LANGUAGE Program**
      1. **Program**

**The Phonon** LANGUAGE program is composed of one special function: “mainAudio.” (Method name) defined as follows.

**<program>** → **mainAudio( ^ args… ^ ).** {

^ … ^

}

* + 1. **DATA**

Data (variables) can be declared anywhere in a function.

**<data\_session>** → **function.()** {

<opt\_varlist\_declarations>

}

**Variable Lists**

The optional variable list declarations is used to define several data type declarations:

**<opt\_varlist\_declarations>** →<varlist\_declarations> | ϵ

**Variable Declarations**

**<varlist\_declarations>** → <varlist\_declaration>

| <varlist\_declarations><varlist\_declaration>

* **PROBLEM DETECTED: Left recursion – SOLVING FOR YOU:**

**New Grammar**

**<varlist\_declarations>** → <varlist\_declaration> <varlist\_declarationsPrime>

**<varlist\_declarationsPrime>** → <varlist\_declaration> <varlist\_declarationsPrime> | ϵ

Each variable declaration can be done as follows:

**<varlist\_declaration>** → <integer\_varlist\_declaration>

| <float\_varlist\_declaration>

| <string\_varlist\_declaration>

* + 1. **Declaration of Lists:**

The variables list declaration is defined here:

**<integer\_varlist\_declaration>** → **int** <integer\_variable\_list>;

**<float\_varlist\_declaration>** → **float** <float\_variable\_list>;

**<string\_varlist\_declaration>** → **string** <string\_variable\_list>;

* + 1. **List of Variables:**

The list of variables is defined here:

**Integers:**

**<integer\_variable\_list>** →<integer\_variable>

| <integer\_variable\_list>, <integer\_variable>

**<integer\_variable>** →VID\_T

**Float-points:**

**<float\_variable\_list>** →<float\_variable>

| <float\_variable\_list>, <float\_variable>

**<float\_variable>** →VID\_T

**Strings:**

**<string\_variable\_list>** →<string\_variable>

| <float\_variable\_list>, <string\_variable>

**<string\_variable>** →VID\_T

* + 1. **CODE session:**

The second part (CODE) is the place we have statements:

**<code\_session>** → **function.()** {

<opt\_statements>

}

**Optional Statements:**

**<opt\_statements>** → <statements> | ϵ

* + 1. **Statements**

**<statements>** → <statement> | <statements> <statement>

* 1. **Statement**

**<statement>** → <assignment statement> | <selection statement> | <iteration statement>

| <input statement> | <output statement>

* + 1. **Assignment Statement**

**<assignment statement>** → <assignment expression>

* + 1. **Assignment Expression**

**<assignment expression>** → <integer\_variable> = <arithmetic expression>

| <float\_variable> = <arithmetic expression>

| <string\_variable>= <string expression>

* + 1. **Selection Statement (if statement)**

**<selection statement>** → **if** (<conditional expression>)

**then** { <opt\_statements> }

**else** { <opt\_statements> } ;

* + 1. **Iteration Statement (the loop statement)**

**<iteration statement>** → **while** (<conditional expression>)

{ <statements>}

**Variable List:**

**<variable list>** → <variable identifier> | <variable list>,<variable identifier>

**Variable Identifier:**

**<variable identifier>** →<integer\_variable>

| <integer\_variable>

| <string\_variable>

* + 1. **Output Statement**

**<output statement>** → **print.**(<opt\_variable list>);

**Optional Variable List:**

**<opt\_variable list>** →<variable list> | ϵ

* 1. **Expressions**
     1. **Arithmetic Expression**

**<arithmetic expression>** → <additive arithmetic expression> **|** <multiplicative arithmetic expression>

**Additive Arithmetic Expression:**

**<additive arithmetic expression>** →

<additive arithmetic expression> + <multiplicative arithmetic expression>

| <additive arithmetic expression> - <multiplicative arithmetic expression>

| <multiplicative arithmetic expression>

**Multiplicative Arithmetic Expression:**

**<multiplicative arithmetic expression>** →

<multiplicative arithmetic expression> \* <primary arithmetic expression>

| <multiplicative arithmetic expression> / <primary arithmetic expression>

| <primary arithmetic expression>

**Primary Arithmetic Expression:**

**<primary arithmetic expression>** → <integer\_variable>

| <float\_variable>

| RL\_T | IL\_T

| (<arithmetic expression>)

* + 1. **String Expression**

**<string expression>** →

<primary string expression> | <string expression> + <primary string expression>

**Primary String Expression:**

**<primary string expression>** → <string\_variable> | SL\_T

1. SOFIA (from Greek, “Wisdom”) is also the name of the Bulgarian capital city, homeland form prof. Svillen Ranev, professor from Compilers for several years in the Algonquin College. [↑](#footnote-ref-0)