Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Team:

Isaac Ribeiro - Id: 040957075 / Hilary Johnson - Id: 040980317

**Language Name: Phonon**

| **Part**  **1** | **Language User Reference** |
| --- | --- |

**EXPLANATION**

The purpose of this assignment is to invent a new Domain-specific computer language for audio processing.

* 1. **User Manual**

**Element 1: Name / Extension**

Name: **Phonon**

Extension: **.phn**

* Our language borrows syntax and patterns from graphics languages like: HLSL/GLSL
* Phonon code will define a wavefunction; the compiler will then quantize it and produce a .wav file

**Element 2 – Comments**

* Single line: **// This is a comment**
* Muli-line: **/\* This is a comment,**

**this is also part of the comment \*/**

**Element 3 – Keywords**

* while
* if
* else
* return
* void
* out
* In
* float
* int
* bool

**Element 4 – Datatypes**

* float (32/64 bits - defined using compiler arguments)
* int (32-bit; −2147483648 - 2147483647)
* bool (8-bit; 0 - 1 / true - false)

**Element 5 – Variables**

* float testFloat;
* int testInt;
* float testFloat2 = 2.0;
* int testInt2 = 2;

**Element 6 - Commands**

### Assignment/Attribution

* testFloat = 3.0;
* testInt = 2;
* Casting is allowed by declaring the datatype needed to cast followed by the value in parentheses
  + eg.int( 0.9 )
* Math is handled as in HLSL/GLSL/C with basic operators built-in functions such as sin(), saw(), triangle(), and square(); fmod(), exp(), abs()

### Selection

* if / else if / else
* Boolean operations same as C: <, >, <=, >=, &&, ||, !

### Interaction

* C-style loop syntax; while, for

### Input

N/A - user input is not necessary for this DSL. Data should always be hardcoded in Phonon.

### Output

N/A - there is no text output necessary; testing/debugging should always be done by ear (after compilation)

### Functions

* returnType functionName(in inputParam, out outputParam) {  
   // body  
  }
* Parameters are for both input and output as in HLSL, allowing multiple returns and inputs
* Values can be returned via assignment of params marked *out*

**Element 7 – Proper elements**

N/A

| **Part**  **2** | **Examples** |
| --- | --- |

**Option 1: Julia-like**

**Music Note: A Major (sine)**

|  | /\* this is the music note A Major expressed in Phonon with a sine wave \*/  void mainAudio(out float a, in float t) {  // sin() produces the wave shape  a = sin(t \* 440.0);  } |  |
| --- | --- | --- |

**Music Note: A Major (saw)**

|  | /\* this is the music note A Major expressed in Phonon with a saw wave \*/  void mainAudio(out float a, in float t) {  // saw() produces the saw wave shape:  // t = 439.0; a = 439.0;  // t = 440.0; a = 0.0;  a = saw(t \* 440.0);  } |  |
| --- | --- | --- |

**Music Note: A Major Chord**

|  | /\* this is the A Major music chord expressed in Phonon with sine waves \*/  void mainAudio(out float a, in float t) {  float a2 = sin(t \* 110.0);  float e3 = sin(t \* 164.81);  float a3 = sin(t \* 220.0);  float cs3 = sin(t \* 138.59);  float e4 = sin(t \* 329.63);  a = a2 + e3 + a3 + cs3 + e4;  } |  |
| --- | --- | --- |

| **Part**  **3** | **Architectural Aspects** |
| --- | --- |

**Advantages**

Phonon presents a novel way to create and process audio.

Most digital audio processing solutions simulate physical tools and avoid the math of music. Phonon flips this paradigm on its head, allowing users to define audio through code as math expressions.

**Strategy: C Implementation**

Similarly to C, Phonon will be parsed primarily based on context. For example, a datatype (eg. int) may be interpreted as a return type or as a variable type, depending on whether it is inside a block scope.

Function calls will also be parsed based on context - referencing a symbol with subsequent parentheses (eg. sin()) is parsed as a function call, whereas a symbol with no parentheses is considered a variable.

Further, any code inside a function call’s parentheses will be parsed as parameters, with each parameter delimited by commas.

Function declarations will be parsed similarly to function calls, except a return type, argument types/names, and code block will also be expected.

Code blocks, both for function bodies and conditional / loop bodies will be delimited using { and } - just like in C. Any code within these delimiters will be considered as part of the block.

Numeric literals will be parsed based on the presence of a decimal, ie. ‘1’ is parsed as an int and 1.0 is parsed as a float.

**Basic ideas about C implementation**

We imagine that we will need to create several structures to implement Phonon in C. For example, the arguments of a function will need to include whether they are an input or output, which data type they are, and the data itself. All of this information can be encapsulated by a C structure. Functions themselves may also be stored as structures, with each instance containing a function name, return type, code body, and array of parameter structures.

We think memory allocation will be the greatest challenge since many variables of different widths will need to be stored in the stack. We are not sure how to best address this challenge yet and are looking forward to learning how.

**Problems when using C implementation**

As stated above, one challenge we foresee is memory allocation since we will have to store variables of differing type and width. We also think defining the parsing rules could prove a challenge since it will be based on context rather than relying on explicit notation.

**References**

* <https://pages.mtu.edu/~suits/notefreqs.html>
* <https://www.justinguitar.com/guitar-lessons/the-a-chord-bc-112>
* <https://en.wikipedia.org/wiki/Standard_tuning>

Algonquin College

Winter, 2023