C-Major: A Music Composition Language

O'Reilly, Andrew `ajo2119` | Huang, Stephanie `syh2115` | Sun, Jonathan `jys2124` | Tang, Laura `lt2510`

September 30, 2015

1 Introduction

C-Major is an interpreted, functional language for composing music and producing structured, sequential, and concurrent sound. It is meant to be used by programmers seeking to compose and sequence simple music to accompany applications or other activities that occur in a computed context.

2 Motivation

C-Major provides a method of organizing modern western music, which consists of a sequence of pitch or frequency values that vary in duration and are often played concurrently. Since modern popular music consists extensively of reused and repeated components, C-Major lends itself to the problem of writing or cutting and pasting the same sequence repeatedly when using conventional notation or audio production software by condensing pitch sequences into reusable components. It provides an intuitive functional interface by which composers can create collections of pitch sequences and modify them according to their desires or the requirements of the composition they are producing.

3 A Simple Deconstruction of Music Composition

A phrase of music consists of several notes strung together. Each note consists of two major values: pitch and duration. The pitch denotes the frequency of the sound, and the duration indicates the amount of time the note is sung. Composition can then become two separate activities: one, a sequence of pitches ignoring rhythm, and two, the underlying rhythm, ignoring pitches. Combining these two lines of values will yield a complete phrase of music.

4 Lexical Conventions

C-major aims to become a functional programming language with clean and simple syntax modeled after languages like Python. It features immutable memory, no global variables, and it has I/O. A typical program is primarily comprised of constant definitions and functions. C-Major files will have the extension .cmaj.

4.1 Comments

Comments will be the same as in C. The characters '/*' introduce a comment, which terminates with the characters '*/'. Comments do not nest, and they do not occur within string or character literals. Single line comments can be initiated with characters '//'.

4.2 Line Structure, Scoping & Delimiting

The end of a logical line is represented by the semicolon token, just like in C. Scoping and delimiting can be determined by curly brackets, like in C, but optional as long as lines are syntactically correct.

4.3 Variables - Identifiers

Identifiers behave just like in C or Java, and are used to label functions and objects. An identifier is composed strictly of upper and lowercase characters, digits, and the underscore character. Case is significant. Scoping will be modeled after Python scoping.

4.4 Types

The following are the basic types of the language.

1. `int, double, boolean`
2. `array/list`
3. `string`
4. `tuple` - Tuples behave like tuples in Python
5. `pitch` - A pitch consists of the dollar sign character followed by a letter, then an optional integer to specify octave (this denotes frequency).

```
play $C;
```

6. `duration` - A duration is a tuple of two integers. Corresponding to elementary music theory, 1st num : 2nd num represents the fraction of a whole note duration.

```
def quarter_note = (1,4); // type duration
def half_note = (1,2);
```

7. `note` - A note is an object with a pitch and a duration.

4.5 Derived Types

Derived types consist of ordered collections and may be constructed by statically placing a set of fundamental types into a collection from another collection, or by returning objects of a given type from a function.

4.6 Operators

Follow standard orders operation (left to right) and mimic AINSI C.

1. Logical negation: `!`
2. Multiplicative: `* / and %`
3. Additive: `+` −
4. Relational: > < >= <=
5. Equality operators: `== !=`
6. Logical AND OR operators: `&&,` ||
7. Assignment Expressions: `= *= /= %= += -=`

4.7 Control Flow

1. Selection statements: `if, else`
2. Iteration statements: `for, while`

```
if ( aIsValid )
  doA();
if ( bIsValid && cIsValid) {
  doB();
  doC();
}
else
  doD();
for element in myArr
  print element;
```

4.8 Functions

Functions follow python convention are intuitive and simple to make and use. Similar to OCaml, parenthesis for function parameters are optional.

```
def gcd (a, b) = {
  while (a != b) {
    if (a > b)
      a -= b;
    else
      b -= a;
  }
  return a;
}
```

5 Sample Code

Here is an example of a simple melody one can write with C-Major.

```
// Twinkle Twinkle

def quarter_note = (1,4); // type duration
def half_note = (1,2);
def rhythm1 = quarter_note*4 + half_note;
def phrase1 = $C*2 + $G*2 + $A*2 + $G;
def phrase2 = $F*2 + $E*2 + $D*2 + $C;
def full_phrase = phrase1 + phrase2
  + (raise phrase2)*2 // up above..like a diamond..
  + phrase1 + phrase2;

def twinkle = mash rhythm1 full_phrase;

play twinkle;
```