Project Report

A programming language for exploring and creating music

Kevin Chen `kxc2103@columbia.edu`
Brian Kim `bck2116@columbia.edu`
Edward Li `el2724@columbia.edu`

December 22, 2015

Contents

1 Introduction

We created a language optimized for exploring and composing music. The syntax facilitates writing snippets of music. Standard library functions transform and combine these snippets, allowing the programmer to take advantage of the repetition found in many songs. The song can then be written to an audio file (WAV).

Our language revolves around the concept of tracks. A track is a variable length sequence of chords, each with a list of pitches and a length, plus metadata such as key signature, tempo, time signature. Tracks are the building blocks of a composition, because they allow the user to specify a few snippets of music, and then concatenate, overlay, transform, or otherwise reuse them while creating a song.

♫# is also a modern language that provides type inference, static typing, low-overhead memory safety, and a clean, readable syntax. Users can simply concentrate on writing a great song, and let the compiler take care of the rest.

1.1 Related Work

At the beginning of this project, we found SoftSynth's Hierarchical Music Specification Language (HMSL),[1] a language with similar goals distributed from 1986 to 1996. It included lots of music-related syntactic sugar:[2]

```
\ Define a simple melody as a motif.
: MOTIF1  ( -- , this is a comment )
       C3 D G E4 A3 G
;
\ Play that motif as quarter notes then sixteenths
PLAYNOW   1/4 motif1  1/16 motif1
```

Written in and inspired by Forth, HMSL was also Turing-complete (not a configuration file). It was possible to write loops and conditionals:

```
\ Play random notes in one octave range
16 0  \ set up loop
DO  12 choose \ random over 1 octave
    60 + \ above middle C
    note \ play it
LOOP
;
```

The language used fractions to represent duration: for example, `1/4` was a quarter note. Equivalently, programmers could use constants such as `Q`. The period `.` operator multiplied a duration by 1.5: `Q.` was a dotted quarter note.

Once notes and durations were specified, the programmer could apply functions to them, such as `SET.CRESCENDO` and `HUMANIZE!` (add random errors).[3]

[1] HMSL homepage: `http://www.softsynth.com/hmsl/index.php`

[2] Code examples adapted from HMSL Overview. `http://www.softsynth.com/hmsl/hmsl_details.php`

[3] HMSL manual, Chapter 16, Score Entry System. `http://www.softsynth.com/hmsl/docs/original/hman/index.php`

We liked the functional aspects and rich standard library of HMSL, but wanted a lightweight syntax.[4] We also wanted to specify pitches in a culturally agnostic way, rather than tying our language to the seven-note Western scale.

[4]The same company also made Java Music Specification Language (JMSL). However, it appears to be a Java library, rather than a language. And the syntax is even more verbose — it takes nearly 100 lines to play a single note: `http://www.algomusic.com/jmsl/examples_with_source/jmslexamples/simple/InstrumentOnOff.java`

2 Tutorial

2.1 Hello World

To compile the compiler and check that it is working correctly, run `install_dependencies.sh` and then `make test` in the `src/` directory.

Here's a simple program in our language. It calls the function `PrintEndline` on the string `"Hello world!"`.

```
// hello_world.nh
PrintEndline "Hello world!"
```

To compile and run the program:

```
$ ./nhc.native -c hello_world.nh -o hello
$ ./hello
Hello world!
```

2.2 Making Music

Here, in seven short lines, we are able to compose "Twinkle, Twinkle Little Star" and its harmony line, and produce a playable audio file from it.

```
1  intro = quarter:[ 1 1 5 5 6 6 ] . half:5
2  chorus = Rhythms intro : [ 4 4 3 3 2 2 1 ]
3  bridge = Relative 1 chorus
4  melody = intro . chorus . bridge . bridge . intro . chorus
5  harmony = Relative 2 twinkle
6  twinkle = Parallel { melody harmony }
7  Render twinkle "twinkle.wav"
```

In line 1, the melody and rhythm of the first line of the song are assigned to `intro`. Pitches are specified as numbers relative to the key signature.[5] Rhythms can be specified using constants such as `quarter` and `half`, or with floating point numbers where 1.0 is a whole note.[6] Notice that the user experience is clean and simple — "these are all quarter notes, and that's a half note."

```
intro = q:[ 1 1 5 5 6 6 ] . h:5
```

The square brackets in lines 1 and 2 are syntactic sugar for the musical list constructor. The `quarter:[...]` applies the quarter note length to each note in the list, and returns a track. Similarly, `h:5` describes a half note of pitch 5. Note that `quarter` and `q` are equivalent.[7]

Function calls do not require parentheses or commas. For example, in line 2, the standard library function `Rhythms` receives the argument `intro`. `Rhythms` takes `intro` and returns an array of the notes' lengths.

[5]Since we have not specified a key signature, the default of C major is used. Therefore, pitch 1 corresponds to C, 2 corresponds to D, 3 corresponds to E, etc.

[6]The default time signature is 4/4, and the default tempo is 120 bpm.

[7]Other predefined note durations include eighths, wholes, and triplets. For more information about these constants, see section 3.8

Then, the colon `:` operator zips the returned rhythm to the notes. We've created a second track, `chorus`, that reuses the rhythm from `intro`. The library frees us from repeating ourselves — we want the same rhythm as before, but with these notes instead.

```
chorus = Rhythms intro : [ 4 4 3 3 2 2 1 ]
```

In line 3, the standard library function `Relative` shifts `chorus` up by one pitch, and assigns the returned value to `bridge`. Here, our language highlights the relationship between the musical phrases of the children's classic.

The standard library is full of abstractions for manipulating tracks in different ways. The result is that we are freed from dealing with the low-level considerations of what notes go where, which notes need to be sharp or flat, and so on. Instead, we can now focus on how the musical phrases interact.

```
bridge = Relative 1 chorus
```

Line 4 creates the song by concatenating the pieces we have made so far:

```
melody = intro . chorus . bridge . bridge . intro . chorus
```

Line 5 creates a harmony to the melody in a single line by using `Relative` to shift `melody` up by two pitches. Recall that we already used `Relative` in this composition for a completely different context. The tools provided by the standard library are powerfully versatile.

```
harmony = Relative 2 twinkle
```

Line 6 aligns the `melody` and `harmony` tracks in a song by using `Parallel`, and assigns the song to `twinkle`. Note that here, we have used the general list constructor `{...}` to create a list of two tracks.

```
twinkle = Parallel { melody harmony }
```

The last line creates an audio file of our song named `twinkle.mp3`.

```
Render twinkle "twinkle.wav"
```

2.3 Advanced Music Composition Features

This is a tutorial for the children's classic "Row, Row, Row Your Boat" in three different styles.

```
// Row, Row, Row Your Boat
de = 1.5*e
se = e/2.0
rhythm = [ q q de se q de se de se h ] . Repeat 12 [ triplet ] .\
    [ de se de se h ]
tune = [ 1 1 1 2 3 3 2 3 4 5 ] . Repeat 3 [ 1@1 ] . Repeat 3 [ 5 ] .\
    Repeat 3 [ 3 ] . Repeat 3 [ 1 ] . [ 5 4 3 2 1 ]

row_your_boat = rhythm : tune
Render (Parallel { row_your_boat }) "row_your_boat.wav"

```

```

// Row, Row, Row Alone (Sad Boat)
key_sig = c_minor
tempo = 80
// the key signature is grabbed when the track is created!
sad_boat = rhythm : tune
Render (Parallel { sad_boat }) "row_alone.wav"


// Roll, Roll, Roll Ya Rims (Gangsta Boat)
key_sig = f_minor
time_sig = two_two
// make it sound dirty and dangerous
swag_boat = Octave (-1) (rhythm : tune)

// add a background track
sick_beat = Repeat 2 [ e e e (e/2.0) (e/2.0) ] . [ e e de se e e q ]
sick_tune = @1[ 1@1  5 6 6 5 4 4 5  1@1  1@1  1@1  5 6 5 4 4 ~ ]
sick_accomp = sick_beat : sick_tune . \
            EndWith (e:[ 7  1@1  ~ ]) (sick_beat : sick_tune)

gangsta_boat = Parallel { swag_boat sick_accomp }
gangsta_boat$volumes = { 1.0 0.35 }
Render gangsta_boat "roll_ya_rims.wav"
```

We begin similarly to Twinkle, by specifying rhythms and pitches. The first version is rendered as `row_your_boat.wav`. Notice that we can define rhythms and pitches separately for easy reuse!

```
// Row, Row, Row Your Boat
de = 1.5*e
se = e/2.0
rhythm = [ q q de se q de se de se h ] . Repeat 12 [ triplet ] .\
    [ de se de se h ]
tune = [ 1 1 1 2 3 3 2 3 4 5 ] . Repeat 3 [ 1@1 ] . Repeat 3 [ 5 ] .\
    Repeat 3 [ 3 ] . Repeat 3 [ 1 ] . [ 5 4 3 2 1 ]

row_your_boat = rhythm : tune
Render (Parallel { row_your_boat }) "row_your_boat.wav"
```

In the second version, all that is needed to change the key signature, time signature, or tempo is to set the standard library globals. Here we set the key signature and tempo to give the piece a sadder feeling.

```
// Row, Row, Row Alone (Sad Boat)
key_sig = c_minor
tempo = 80
```

Since the key signature, tempo, and time signature are saved into the track on creation, all we have to do is re-zip our rhythms and pitches to create a new track with the new settings. Note how easy it is to transpose an entire piece! Our second version is appropriately rendered as `row_alone.wav`.

```
// the key signature is grabbed when the track is created!
sad_boat = rhythm : tune
Render (Parallel { sad_boat }) "row_alone.wav"
```

In the third version, we shift the melody down an octave for an absolutely swag feeling. To swag up the classic children's tune, we also changed the key and time signature.

```
key_sig = f_minor
time_sig = two_two
// make it sound dirty and dangerous
swag_boat = Octave (-1) (rhythm : tune)
```

We also add a background track, as all swaggalicious pieces must have. Notice in line 29, we use the octave operator on an entire music list. Also notice in line 31, we use the `EndWith` function. This is another example of an abstraction the standard library provides — "I want the same track, except replace the end of it with this other track."

```
// add a background track
sick_beat = Repeat 2 [ e e e (e/2.0) (e/2.0) ] . [ e e de se e e q ]
sick_tune = @1[ 1@1  5 6 6 5 4 4 5  1@1  1@1  1@1  5 6 5 4 4 ~ ]
sick_accomp = sick_beat : sick_tune . \
            EndWith (e:[ 7  1@1  ~ ]) (sick_beat : sick_tune)
```

Once again, with just a few more lines of ♫# code, we are able to create a completely different piece. We render this final spin on the children's classic as `roll_ya_rims.wav`. Notice that we can also adjust the volume mix ratios of the tracks. That was mad easy, yo.

```
gangsta_boat = Parallel { swag_boat sick_accomp }
gangsta_boat$volumes = { 1.0 0.35 }
Render gangsta_boat "roll_ya_rims.wav"
```

2.4 Defining Functions

We've seen how to apply functions provided by the standard library. Now we'll define our own functions using the `fun` keyword:

```
fun Concatenate x y = x . " " . y
PrintEndline (Concatenate "Hello" "World") // => "Hello World"
```

Just like we didn't have to specify types when declaring variables, we also don't have to specify the argument types or return values of functions. Since everything in ♫# has a return value, functions return the value of the last expression executed.

2.5 Control Flow

♫# also supports conditionals and loops. For example, the function below returns the integers on the interval [`low`, `high`):

```
fun Range low high = (
  if high <= low then
    int{}
  else
    { low } . Range (low + 1) high
)
```

`Range` recursively calls itself and eventually returns an empty array, appending all of the intermediate arrays afterwards. Since the function body spans multiple lines, we need to wrap it in parentheses.

We can test our function by using a loop to iterate through the returned array:

```
for i in Range 0 10 do (
  PrintInt i
  Print " "
)
```

The loop sets `i` to each item in the array and executes the body. The program prints the range $[0, 10)$ to the console:

```
0 1 2 3 4 5 6 7 8 9
```

Here's the entire program:

```
fun Range low high = (
  if high <= low then
    int{}
  else
    { low } . Range (low + 1) high
)

for i in Range 0 10 do (
  PrintInt i
  Print " "
)
```

3 Language Reference Manual

3.1 Types and Literals

3.1.1 Primitive Types

Unit (`unit`): A unit literal is specified as `()`. The unit literal is the only value that the unit type has.

Boolean (`bool`): May be `true` or `false`.

Integer (`int`): A literal such as `1564` is a 64-bit signed integer.

Floating point (`float`): A floating point literal has a decimal part `156.4`, or an exponent part `2e-4`, or both. These are IEEE 754 double-precision (64-bit) numbers.

String (`string`): A sequence of ASCII characters. String literals are enclosed in double quotes, with special characters escaped with a backslash `\`.

```
"I am an alpaca, and I say \"Pikachu\" all the time.\n"
```

The supported escape sequences are:

`\n`	newline	`\r`	carriage return
`\t`	horizontal tab	`\v`	vertical tab
`\\`	backslash	`\"`	double quote

Pitch (`pitch`): Pitches are written as *note*`@`*octave-offset* — both `int`s. For example, `3@1` is the third note of the current key signature, at one octave above the octave where A is 440 Hz. (If the octave is negative, parentheses must be used: `3@(-1)`.)

3.1.2 Arrays

Array literals are a sequence of literals enclosed in curly braces. The items are not separated by commas or semicolons. For example, these are valid arrays:

```
{ 1 2 3 }
{ "red" "orange" "yellow" "green" "blue" "violet" }
```

Arrays are strongly typed — all elements of an array must be of the same type:

```
{ 1 2 "3" } // Type error
{ 1 2 3.0 } // Type error
```

3.1.3 Empty Arrays

Some situations require empty arrays, which may cause the type of the array to be ambiguous. In this case, the compiler requires prepending the type name to the array literal: `string{}`.

3.1.4 Music Array Syntax

Musical array literals are enclosed by square brackets instead of curly braces. This syntax can only contain chords and durations.

This eliminates ambiguity between pitches and integers: `{ 1 2 3 }` is interpreted as an array of `int`, while `[ 1 2 3 ]` is an array of `chord` (where each chord only happens to have one pitch).

It also eliminates the ambiguity between durations and floats: when float literals appear in musical array literals, they are always interpreted as an array of duration.

```
[ 6,7,8  9 10 ] // 6,7,8 represents a chord: the notes are played simultaneously
[ 0.25 1.0 1.5 ] // a rhythm of quarter note, whole note, dotted whole
```

3.2 Chords

A chord is a collection of pitches. To create a chord, separate the pitch literals with commas: `1@1,3@1,5@1`. Syntactic sugar for a rest (empty chord) is a tilde `~`.

3.3 User-defined Types

The `type` keyword creates a user-defined type, which may consist of primitive types and other user-defined types. The definition must contain default values for each member: the type of each member is inferred from the default values. Default values must not contain function calls.

```
type person = {
  name = ""
  age = 0
  favorite_ice_cream = string{}
}
```

To create a new instance of a user-defined type, we use `init` *typename*.[8] Member variables are mutable and can be accessed using the `$` operator.

```
friend = init person
friend$name = "Stephen Edwards"
friend$age = 21
friend$favorite_ice_cream = { "durian" "Taiwanese fish sandwich" }
```

Equivalently, we can override the default values at initialization. This syntax is also less verbose:

```
friend = init person {
  name = "Stephen Edwards"
  age = 21
  favorite_ice_cream = { "durian" "Taiwanese fish sandwich" }
}
```

[8]The keywords `beget` and `bringintobeing` are accepted as replacements for `init`.

3.4 Operators and Expressions

3.4.1 Identifiers

Identifiers are sequences of letters, digits, and underscores. Additionally, function identifiers must begin with an uppercase letter, while type and variable identifiers must begin with a lowercase letter or underscore.

Valid function names: `Assert`, `Merge_sort`, `Duplicate3x`
Invalid function names: `_Assert`, `mergeSort`, `3X_Duplicate`

Valid variable and type names: `_count`, `input_file_2`, `favoriteNumber`
Invalid variable and type names: `Count`, `2nd_input_file`, `favorite-number`

3.4.2 Variables and Assignment

The `=` operator is used to assign the value of an expression to an identifier. It returns a unit. Additionally, assignment is non-associative, so chaining assignments is a syntax error.

```
my_jelly_beans = 1000
my_jelly_beans = my_jelly_beans + 60 // Bought some more jelly beans
i = j = 0 // => Syntax error
```

The first line implicitly declares a new variable, since the identifier `my_jelly_beans` has not appeared previously in the program. Thanks to type inference, we don't have to specify the type.

To declare a constant, prefix the identifier with the `const` keyword:

```
const planets_count = 9
planets_count = 8 // => Compile-time error
```

3.4.3 Arithmetic Operators

The arithmetic operators are `+`, `-`, `*`, `/`, and modulus `%`.

The unary `-` operator has the highest precedence, followed by the binary `*`, `/`, and `%` operators, followed by the binary `+` and `-` operators. All arithmetic operators are left-associative.

Although we do not have a separate set of operators for floating-point arithmetic, arithmetic operators may only be applied to operands of the same type — there is no automatic promotion of `int` to `float`. For example, `1 + 1.0` is a type error.

3.4.4 Logical and Relational Operators

Relational operators are `<`, `<=`, `>`, `>=`, which have the same precedence. The equality operators `==` and `!=` are below them in precedence, then `&&`, then `||`.

In equality comparison, primitives are compared by value. Collections and user-defined types are compared structurally: each member is compared by value. For example, the following boolean expressions are equivalent:

```
type stringnum = {
  s = "zero"
  n = 0
}
a = init stringnum
b = init stringnum
PrintBool (a.s == b.s && a.n == b.n) // => "true"
PrintBool (a == b) // => "true"
```

The negation operator `!` inverts true to false and vice versa. Unlike C and C++, it does not convert non-zero values to zero: negation may only be applied to `bool` operands.

3.4.5 Array Operators

3.4.6 Array Access

The *array-identifier*`.(`*int-expression*`)` operator access an element of an array. Arrays are immutable, so assigning to elements using this syntax is not allowed.

```
arr = { 0 1 2 3 4 }
PrintInt arr.(2) // => "2"
arr.(2) = 5 // Compile-time error
```

3.4.7 Concatenation

The period `.` binary operator[9] concatenates arrays, pitches, or strings. It is left-associative.

```
instruments = { "violin" }
instruments = { "piano" } . instruments // => { "piano" "violin" }
repl = "u" . "top" // => "utop"
favorite_numbers = { 3 9 } . { "twenty-seven" } // Type error
```

3.4.8 Sharp and Flat

Sharp `#` and flat `b` are unary operators that respectively increase and decrease their operand by a half step.

```
[ 5# ] == [ (init pitch { rank = 5; offset = 1 }) ] // => true
[ 3b ] == [ (init pitch { rank = 3; offset = -1 }) ] // => true
```

[9]♫# incorporates ideas from many well designed, highly respected languages. Using the period character for concatenation was an idea we took from PHP.

3.4.9 Musical Zip

The colon operator zips durations and pitches into a track: *rhythm-expr*`:`*pitch-expr*. The rhythm may be a single duration or an array of durations, and the pitch may be a single pitch or an array of pitches. Zipping two arrays of different lengths causes an exception at runtime.

For more information about tracks, see section 3.8.2.

```
same_duration = quarter : [1 2 3 6 7]
same_note = [quarter half half quarter half half] : 1,3,5
everything_changes = [quarter half] : [1 2@1]
short_track = quarter : 2
short_rest = quarter : ~
```

3.5 Control Flow

All expressions in ♫# have return types, including control structures.

3.5.1 Conditionals

There are two forms of conditional expressions in our language:[10]

`if` *boolean-expression* `then` *expression* `else` *expression*
`be` *expression* `unless` *boolean-expression* `inwhichcase` *expression*

```
greeting = be "Hello" unless location == "Texas" inwhichcase "Howdy"
if audience_size <= 7 * 1000 * 1000 * 1000 then
  PrintEndline (greeting . " world")
else
  PrintEndline (greeting . " universe")
```

On line 1, we can assign the conditional to a variable. Its value is the value of the last expression in the branch executed.

This also means both outcomes of the condition must be handled: each `if` must have an `else`, and each `be` must have an `inwhichcase`. Conveniently, it also encourages programmers to code more defensively, leading to better code.[11]

3.5.2 For Loop

For loops only iterate over arrays:[12]

`for` *identifier* `in` *array-expression* `do` *expression*

The for loop evaluates the expression for each item in the array, with the identifier assigned to the current array item. The return value of a for loop is unit.

[10]The `be`–`unless`–`inwhichcase` conditional is a revolutionary new language construct we are introducing. Because it provides an easy-to-use way for programmers to spice up their code, we consider it an essential feature of our language.

[11]It also works around the dangling else problem.

[12]Conveniently, there is a standard library function `Range` that generates arrays of ranges. See section 3.8.4.

We do not provide break or continue. Algorithms that require these should be rewritten as tail-recursive functions.

3.5.3 Exceptions

The `throw` keyword throws an exception. If uncaught, the exception is printed and the program exits. Currently, there is no way to catch exceptions.

Because `throw` returns unit, using it in a conditional requires providing a default value to pass the type checker — otherwise, the two branches won't have same type.[13] For example:

```
message = if everything_ok then "Succeeded!" else (throw "Failed!"; "")
```

3.6 Program Structure

3.6.1 Comments

Our language allows single-line comments `//` and nested multi-line comments `/* ... */`.

```
// I'm a single line comment.
/* I am a
   multiline /* nested */ comment. */
```

3.6.2 Includes

All programs must begin with includes, if any. Includes are specified in the following format:

`include` *module-name*.

Including a file dumps all of its functions, types, and globals into the current file. (The standard library is implicitly included at the beginning of each file.) Additionally, it runs any top-level expressions in that library.

The compiler keeps track of which files have been included, so each file is only included once — even if there is a circular include.

```
include phonebook
// Now we can use types, variables, and functions from phonebook
sedwards = init person { name = "Stephen Edwards" }
database = CreatePhonebook "My Columbia Friends" { sedwards }
```

[13] In OCaml, `failwith` has the type signature `string -> 'a`, allowing it to be used in any context. Unfortunately, ♫# does not have the concept of an "any type."

3.6.3 Functions

Functions are defined using the `fun` keyword.[14]

`fun` *Function-identifier arg-identifier-1 . . . arg-identifier-N* `=` *expression*

They can be defined anywhere in the top level of the program, and do not have to be defined before they are called. If there are multiple definitions, the last definition is used throughout the program.

```
Sum 1 2 // => 3
fun Sum x y = x + y
```

Functions are implicitly templated.[15] That means types are checked when the function is called, rather than when it is instantiated. In the example above, we could've passed in four `float` values instead, since the operator `+` is defined on `float`.

The arguments to a function are always passed by value, including collections and user-defined types. Mutating an argument within the caller does not affect the callee's copy.

3.6.4 Foreign Function Interface

The `extern` keyword declares a C++ function header:

`extern "`*header*`" "`*namespace*`" "`*cpp-name*`" fun` *new-name argtype1 . . . argtypeN* `->` *ret-type*

For example, this brings in the `pow` function:

```
extern "cmath" "std" "pow" fun Pow float float -> float
Pow 2.0 4.0 // => 16.0
```

It's important to specify the function's type signature correctly, or the program will not compile. Types are mapped as follows:

♫#		C++
`unit`	↔	`unit_t` (`support.hpp`)
`bool`	↔	`bool`
`int`	↔	`int64_t`
`float`	↔	`double`
`string`	↔	`std::string`
`type{}`	↔	`std::vector<type>`

Foreign functions may be mapped to the same name in ♫# as long as the arguments' types are different. However, foreign and ♫# functions may not share any names.

3.6.5 Scoping

Scoping works naturally. The outermost scope is the whole program. Function definitions create their own scope, which must be enclosed in parentheses if the function has more than one line. Code constructs related to control flow (conditionals and for loops) will create a local scope as well.

[14]We chose `fun` because programs written in our language should be fun!

[15]Implicit templating makes type inference easier to implement. For more information about type inference implementation, see section 5.5.1.

However, there is no implicit declaration within these scopes: if a name is defined in a higher scope, assigning to that name will mutate the original variable rather than declaring a new one.

```
a = 5
b = 6
c = d // Error: d is not defined yet
if a == 5 then (
  a = 6
  d = 7
)
else
  a = 4
PrintInt a // => 6
c = d // Error: d is no longer defined
```

3.6.6 Multi-line Expressions

The line continuation character is backslash `\`. Lines are separated by a newline or semicolon. Multiple statements within the scope of a code construct must be enclosed within parentheses:

```
x = 6
// Multi-line expression
y = 4 + 5 + \
  6 + 7
if x == 5 then
  y = 5
else ( // Multiple expressions within parentheses
  x = 0; z = 7 // Semicolon separates expressions on the same line
  y = 0
)
```

3.7 Standard Library

3.7.1 Settings

Every composition needs a key signature, time signature, and tempo. In our language, we represent these settings as global variables declared in the standard library: `key_sig`, `time_sig`, and `tempo`.

These setting are applied to tracks at construction, so changing them affects all future tracks in the song. The defaults are shown below:

```
// Type that specifies settings of a composition.
key_sig = c_major // Defined in standard library
time_sig = init time_signature
tempo = 120
```

3.7.2 Time Signature

Time signature is represented as a type named `time_signature`. This type contains two values corresponding to the upper and lower half of the time signature:

```
type time_signature = {
  upper = 4
  lower = 4
}
```

3.7.3 Tempo

Tempo is an `int` specifying the beats per minute. The default value is 120 bpm.

3.8 Rhythm Constants

Commonly used beats are `float` constants defined in the standard library for convenience. For example, typing `quarter` or `q` is the same as `0.25`:

`e` or `eighth`	`0.125`
`q` or `quarter`	`0.25`
`h` or `half`	`0.5`
`w` or `whole`	`1.0`
`t` or `triplet`	`0.25 / 3.0`

Using `float` values for note durations also allows us to specify more fine-grained durations with the arithmetic operators:

```
q * 1.5 // => Dotted quarter note
[ (h-e) e h ] // => Syncopated rhythm of 3/8 1/8 1/2 notes
```

3.8.1 Key Signature

Pitches are represented as indices into the key signature so the language can be key-signature agnostic. However, to create audio, we have to specify the mapping between these indices and frequencies. This is the key signature.

For example, the Western scales are mapped as:

```
const c_major = init key_signature {
  scale = { 261.63 293.66 329.63 349.23 392.00 440.00 493.88 }
}
const c_minor = init key_signature {
  scale = { 261.63 293.66 311.13 349.23 392.00 415.30 466.16 }
}
```

Pentatonic, Hexatonic, and Heptatonic scales are defined in a similar manner. However, trying to access a note outside of the scale will result in an exception:

```
key_signature = c_major_pent // 5-note scale
pitches = [ 1 2 3 4 5 ] . @1[ 1 2 ] // OK to access the next octave
pitches = [ 1 2 3 4 5 6 7 ] // => Exception
```

3.8.2 Tracks

Tracks represent musical phrases. They consist of a sequence of chords, and their durations. They also contain the key signature, time signature, and tempo — these values are copied from the globals declared in the standard library `key_sig`, `time_sig`, `tempo` when the track is created.

```
type track = {
  key_sig = init key_signature
  time_sig = init time_signature
  tempo = 120
  chords = chord{}
  durations = float{}
  volume = 1.0
}
```

New tracks are created when an array of chords is zipped with an array of floats (note durations):

```
my_track = [ quarter quarter half ] : [ 5 6 7 ]
```

New tracks are also created when two old tracks are concatenated:

```
new_track = first_track . second_track
```

Note that concatenating two tracks of different key signature, time signature, or tempo causes an exception at runtime.

3.8.3 Songs

A song is a collection of tracks:

```
type song = {
  tracks = track{}{}
  volumes = float{}
}
```

An array of tracks represents tracks to be played sequentially. The array of all these track arrays represents parts to be played concurrently. A song also contains the volume mix ratios for each of these tracks. Many standard library functions create and mix songs, such as `Parallel`:

```
my_song = Parallel { track_1 track_2 track_N }
```

3.8.4 Function Listing

`Min x y`
Returns the minimum of the two elements.

`Max x y`
Returns the maximum of the two elements.

`ReverseList list`
Returns an array with all the elements of `list` reversed.

`IsMember list elem`
Returns true if `list` contains `elem`.

`Range low high`
Returns an array of all integers in the interval [`low`, `high`).

`Size list`
Returns the number of elements in `list`.

`SameList elem iter`
Returns a list with `elem` repeated `iter` number of times.

`Reverse track`
Returns the track with all the chords of `track` played in reverse order.

`Render filename song`
Creates a WAV file of the song.

`Print s`, `PrintEndline s`, `PrintInt i`, `PrintFloat f`, `PrintBool b`, `PrintChord chord`, `PrintRhythms rhythms`
Prints the argument to standard out.

`Exit c`
Exit the program with the specified exit code. If there is no call to `Exit` at the end of the file, `Exit 0` is implicitly called.

`PitchOfInt i`
Returns a pitch with the given rank.

`ChordOfPitch pitch`
Returns a chord containing the given pitch.

`Scale pitch_a pitch_b`
Returns an array of chords representing the scale between `pitch_a` and `pitch_b`.

`Arpeggio chord`
Returns an array of chords representing the arpeggio using the pitches from chord.

`Rhythms track`
Returns the array of note durations of `track`.

`Chords track`
Returns the array of chords of `track`.

`Length track`
Returns the number of beats in `track`.

`Rest`
Returns an empty chord.

`Relative num track`
Returns a track with all the pitches of `track` shifted up by `num`.

`AddChordNum num chord keysig`
Returns a chord with all the pitches of `chord` shifted up by `num` for the given key signature.

`AddPitchNum num pitch keysig`
Returns a pitch shifted up by `num` for the given `pitch` and key signature.

`AddPitchOctave pitch octave`
Returns a pitch shifted up by the given octave.

`Octave num track`
Returns the track shifted by `num` octaves.

`OctaveChordList num chord`
Returns the chord shifted by `num` octaves.

`NormalizePitch pitch keysig`
Returns a whose rank is valid in `keysig` by changing the octave.

`ConcatTracks track_a track_b`
Returns a new track with the tracks arranged sequentially.

`FlatPitch p`
Flats the pitch.

`SharpPitch p`
Sharps the pitch.

`ChordofChords chord_a chord_b`
Returns a chord containing the union of pitches in both chords.

`Extend len track_a`
Returns a track that repeats the given track repeated to fill `len` beats. If `len` is not an even multiple of `Length track_a`, the remainder is padded with rests.

`Repeat times track`
Returns `track` repeated `times` number of times.

`RemoveEnd len tr`
Returns `tr` with the last `len` beats sliced off.

`EndWith tr base_track`
Returns `base_track` with the end replaced by `tr`.

`StartWith tr base_track`
Returns `base_track` with the beginning replaced by `tr`.

`Parallel track_a track_b ...`
Returns a song with the tracks aligned in parallel (to be played concurrently).

`SecondsOfDurations durations timesig tempo`
Converts `durations` converted to seconds based on `timesig` and `tempo`.

`FrequenciesOfChord chord keysig`

Converts `chord` to an array of frequencies based on `keysig`.

`FrequenciesOfChords chords keysig`
Converts each chord in `chords` to an array of frequencies based on `keysig`.

`FrequencyOfPitch pitch keysig`
Converts `pitch` to a frequency based on `keysig`.

`OffsetFrequency frequency offset`
Changes `frequency` for the given `offset` ($\sharp = +1$, $\flat = -1$).

`FrequencyOfChromatic base_frequency chromatic`
Computes the frequency for the given chromatic

`ChromaticOfFrequency base_frequency frequency`
Computes the offset of `frequency` relative to `base_frequency`

3.9 Miscellaneous

3.9.1 Order of Operations

In order of decreasing precedence:

Operators	Description	Associativity
`!`	Logical not	Unary
`-`	Negation	Unary
`*`, `/`, `%`	Multiply, Divide, Modulus	Left
`+`, `-`	Add, Subtract	Left
`<`, `>`, `>=`, `<=`	Comparison operators	Left
`==`, `!=`	Equality operators	Left
`&&`	Logical-and	Left
`||`	Logical-or	Left
`.`	Concatenation	Left
`=`	Assignment	None

Use parentheses `()` to override operator precedence.

3.9.2 Toolchain

The compiler is named `nhc`. It accepts the following command-line arguments:

`-A`	Output internal representation (syntax tree)
`-c` *file*	Compile the specified file
`-o` *file*	Write output to the specified file
`-S`	Output intermediate language representation (C++)
`-v` and `-vv`	Print verbose debugging information

4 Project Plan (Edward)

The group met two to three times per week (immediately after class, and sometimes an additional day of the week). A rough outline of the timeline was set down at the beginning. This included hard deadlines of when each milestone needed to be completed enough such that the next milestone could be started. As the dates got closer, more specific timelines were layed down for completion of features and resolution of bugs and other issues. The first two milestones — language proposal and reference manual — were useful deadlines in that it helped us figure out and limit the exact scope of the project. All language specifications were completed by then, which gave us a good estimate of how many features needed to be implemented. From there, we worked layer by layer, going from scanner, to parser, to ast, to semantically checked ast, to target language ast, to target language code generation. In fact, because we had specified enough of the language by the first milestone - the language proposal - we were able to start working on the first three layers. By the time the LRM was submitted, we had already implemented most of the scanner, parser, and ast. Additionally, because we concurrently wrote the parser and the LRM, we were able to weed out major issues early on, so the final version of our language is essentially a superset of what we specified in our LRM.

After the LRM was submitted, we set up integration testing and required that development of each feature would happen end-to-end: the relevant parts of each layer needed to be implemented, relevant tests needed to be written, and all tests must be passed. Tickets were specified this way, and distributed evenly among the group members. Because we met multiple times per week, we were able to help each other resolve any hardships encountered in the implementation. Ultimately, we were able to complete multiple tickets per week without being bogged down by bugs because of our agile, vertical development and integration process.

We required code reviews for every feature implemented. A pull request would be made, and another member (or two) would review the code and make comments and suggestions. This would be repeated until all members involved were satisfied, at which point the original contributor would push to master. Code reviews ensured high quality in terms of style, and prevented logic errors.

4.1 OCaml Style

- Indent with two spaces.
- Indent to indicate scope.
- Wrap lines at 120 characters.
- Comments are not required, but should be included for confusing or weird-looking code.
- Pattern match as much as possible.
- Use a pipe character `|` with all match cases, including the first one.
- Avoid impossible exceptions as much as possible.
- Be as specific as possible when throwing exceptions.
- Do not repeat code — refactor if possible.
- Use as little mutability as possible.
- Be descriptive and consistent in naming everywhere.
- Use lowercase letters and underscores in naming.
- `open Core.Std` in all files for consistency.

4.2 ♫# Style

- Indent with two spaces.
- Indent to indicate scope.
- Continuation lines should use a hanging indent.
- Wrap lines at 120 characters.
- Array items should be by default be single space separated, with a space separating the open and closing brackets/braces.
- Chords and Octave literals should be double-space separated from the surrounding items, except for open and close brackets/braces.
- The closing brace/bracket/parenthesis on multi-line constructs should be lined up under the first character of the line that starts the multi-line construct.
- Use lowercase letters and underscores for variable names.
- Use PascalCase for function names.
- Avoid extraneous whitespace when using array access, chord construction, zipping non-list literals.
- Surround assignment, boolean, and concat operators with a single space on either side.
- Writing multiple statements on the same line is discouraged.
- All comments should rhyme. Do not put a comment if you can't make it rhyme.

4.3 Project Timeline

- **9/9** — Brainstorm project ideas. Assign roles.
- **9/14** — Decide on a language and brainstorm core concepts of the language, and its syntax.
  9/16 — Hammer out details of core concepts and syntax. Discuss control flow.
- **9/21** — Try to write code in language. Discuss difficulties
  9/23 — Discuss more ideas to smooth out usage.
- **9/28** — Finish Proposal, submit.
  9/30 — Proposal due. Consider and experiment a few target languages and libraries.
- **10/5** — Discuss and set up environment
  10/7 — Work out problems with setting up the same environment on all computers. Implement examples from proposal with target language/libraries.
- **10/12** — Start implementing abstract syntax tree, parser, and scanner to check for potential shift/reduce errors. Hammer out any problems encountered in grammar. **10/14** — Continue implementing ast, parser, and scanner. Have all basic calculator functionality implemented.
- **10/19** — Continue implementing ast, parser, and scanner. Start writing LRM. Parser only: work on barebones features of a programming language implemented (primitive types, arrays, literals, control flow, scoping, function definition, comments).
  10/21 — Continue implementing ast, parser, and scanner. Finish writing LRM. Parser only: work on specific features of the language (pitches, chords, durations, zip, concat, custom types, etc). Finish barebones features of the language.
- **10/26** — Language Reference Manual Due. Continue implementing ast, parser, and scanner.
  10/28 — Continue resolving bugs/conflicts/issues in grammar; Finish all core syntax of the language. Resolve bugs and other potential conflicts/issues.
- **11/2** — (No class.) Start implementing type checker. Finish resolving issues in the grammar. Main compiler: logging module, command line argument parser. Start considering

helper code in the target language.
11/4 — Continue implementing type checker. Start working on translator (target language ast and code gen). Continue integrating with the target language. Implement a few print functions to stdlib.

- **11/9** — Implement working Hello World Demo. Resolve include path issues. Set up test automation.
  11/11 — Continue implementing translator. Resolve Issues with Hello World Demo. Start implementing other basic standard library functions.
- **11/16** — Hello World Demo Due. Implement foreign function interface end-to-end. Add emoji to tester command line output.
  11/18 — Resolve discovered integration issues and bugs. Create C++ AST for code generation. Implement syntax highlighting in Sublime Text.
- **11/23** — Start working on semantic checker (typedef, assignment, initialization, blocks, binary operators). Improve tester to take into account nhc, clang, and test program return values.
  11/25 — (Thanksgiving break.) Implement conditionals and for loops end-to-end. User testing with people[16] who've never seen the language before: discovered and fixed a ton of bugs.
- **11/30** — Implement const and empty lists. Start implementing music functions in standard library.
  12/2 — Implement throw. Add globals and types to function semantic-checking environment. Continue implementing music functions in standard library.
- **12/7** — Implement passing typedefs and arrays across the foreign function interface. Continue implementing other music functions in standard library. Resolve issues in translator.
  12/9 — Continue resolving issues in translator. Continue implementing standard library. Write examples of songs.
- **12/14** — Begin implementing Render standard library function. Implement typedef equality comparison. Continue implementing standard library. Continue writing examples.
  12/16 — Finish implementing Render and other standard library functions. Finish resolving issues in translator. Prepare of live demo.
- **12/21** — Write final report. Write "Stairway to Heaven" example on request from Prof. Edwards. Clean up code (do the high-priority refactoring tech debt tickets).
  12/22 — Final Report Due. Finish editing final report. Drink volumes of alcoholic beverages. At least 1 liter per person.

4.4 Roles and Responsibilities

Kevin Chen	Systems Architect
Brian Kim	Tester
Edward Li	Manager

Because of our vertical integration process, all members touched most parts of the project. Refer to the Compiler Roles table in section 5.9 or the Project Log in section 4.7 for specific contributions.

[16]Two middle school students.

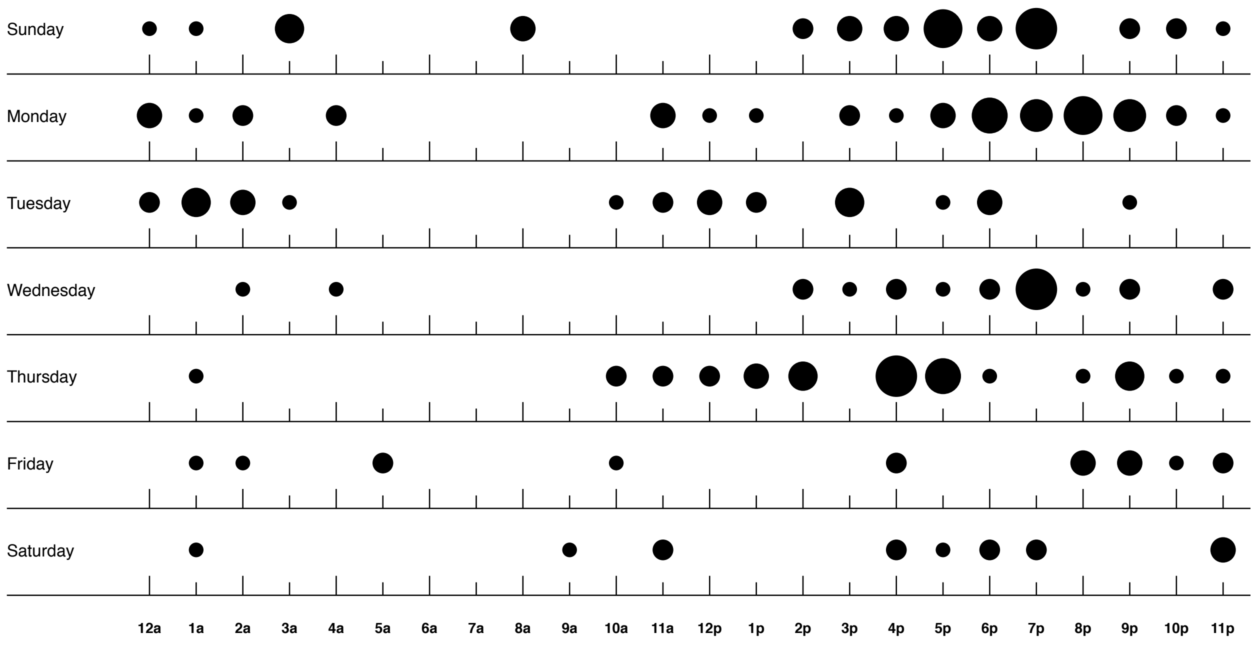
4.5 Development Environment

Language	OCaml 4.02.3 with Core 113.00.00
Build System	Corebuild
Editor	Sublime Text, Vim
REPL	UTop 1.18
Version Control	Git
Bug Tracker	GitHub Issues
Operating System	Ubuntu 15.04, OS X 10.11 "El Capitan"
Lunch & Dinner	Taqueria y Fonda La Mexicana, Szechuan Garden, Ferris Booth

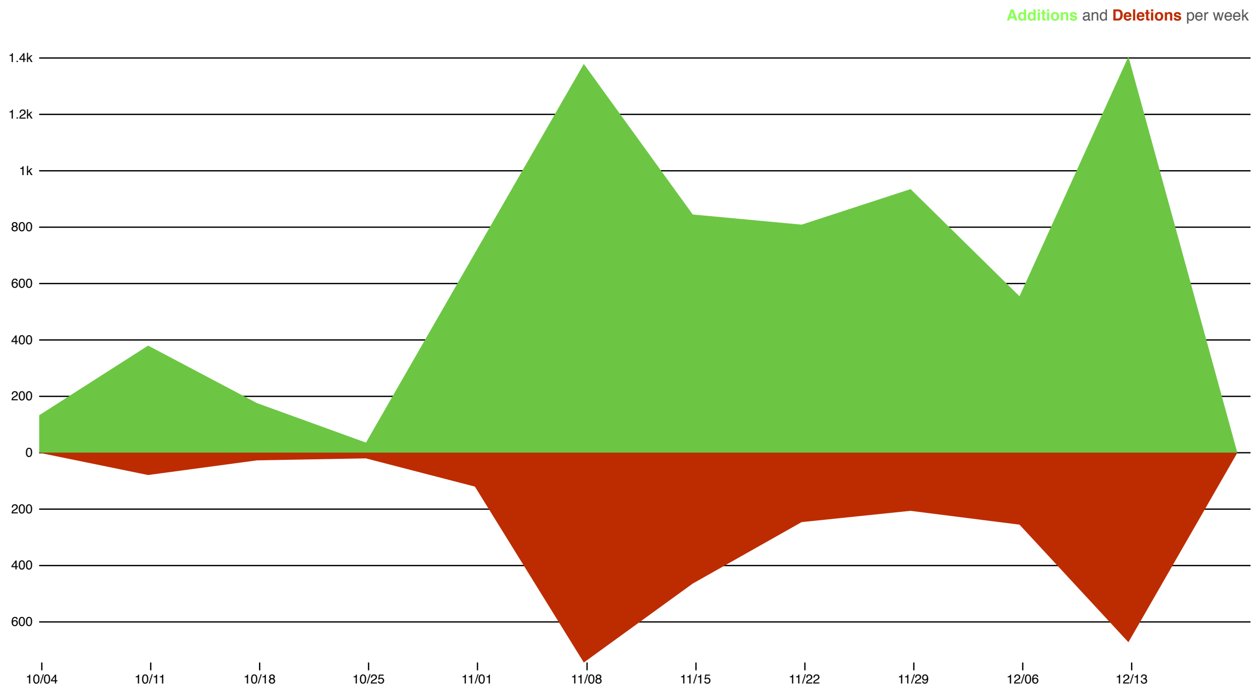
4.6 Version Control Graphs and Statistics

Graphs were produced by GitHub based on commit `b57683f`.

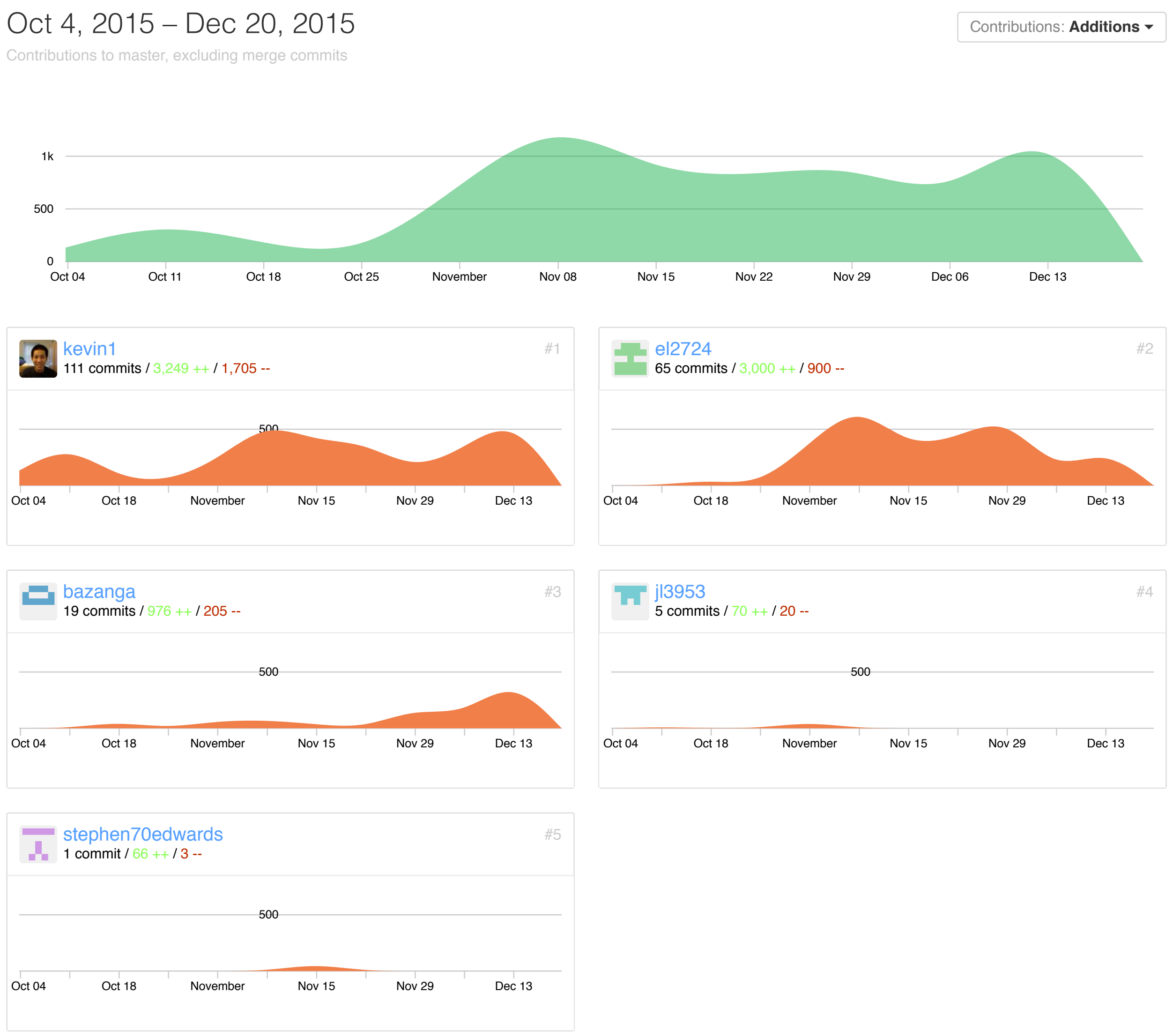
4.6.1 Punch Card



4.6.2 Code Frequency



4.6.3 Contributors



Oct 4, 2015 – Dec 20, 2015

Contributions to master, excluding merge commits

kevin1
111 commits / 3,249 ++ / 1,705 --

el2724
65 commits / 3,000 ++ / 900 --

bazanga
19 commits / 976 ++ / 205 --

jl3953
5 commits / 70 ++ / 20 --

stephen70edwards
1 commit / 66 ++ / 3 --

4.7 Project Log

Commits to our project excluding merge commits.[17] Commits are sorted by commit date, not authoring date, so the dates may appear out of order.

- 2015-12-20 `6beba2b` Kevin Chen Reformat lines with >120 characters.
- 2015-12-20 `30539a2` Kevin Chen tester: Change failure emoji to fire.
- 2015-12-20 `1f4ed22` Edward Li Delete twinkle_simple.nh
- 2015-12-19 `b57683f` Kevin Chen lib: Add PrintInt, PrintFloat, PrintBool functions.
- 2015-12-19 `7a402fb` Kevin Chen lib: Remove unused std_basic.nh.
- 2015-12-19 `58614a7` briank621 Added comments to functions

[17] This project is the only one with a commit "authored" by Prof. Edwards. We used his MicroC AST as the base for our C++ AST.

- 2015-12-19 `8b2f550` briank621 Renamed test functions involving std
- 2015-12-19 `eab6a63` Edward Li added aerosmith example
- 2015-12-19 `d66651d` Edward Li added stairway to heaven example
- 2015-12-19 `b5e1c85` Edward Li flat and sharps now work internally
- 2015-12-19 `9a1bc2a` Edward Li fixed bug in roll ya rims
- 2015-12-18 `8856c1c` Edward Li fixed startwith test
- 2015-12-18 `1d75f3c` Edward Li fixed scale
- 2015-12-18 `f228a01` Edward Li got rid of underscore in name
- 2015-12-17 `c99e156` Kevin Chen support: Render: Suppress STK "creating WAV" message.
- 2015-12-17 `ffcce0c` Kevin Chen cpp_sast: Fix error when calling templated functions.
- 2015-12-17 `80ad254` Kevin Chen cast: Add template arguments in function calls.
- 2015-12-09 `9760122` Kevin Chen optimize: Add constant folding for literals.
- 2015-12-17 `070305e` briank621 Added RemoveEnd functionality and tests
- 2015-12-17 `f28dec6` Edward Li made row boat sound better
- 2015-12-17 `3c54811` briank621 Added negative relative tests
- 2015-12-17 `6762c5e` Kevin Chen style: Update lib types & globals.
- 2015-12-17 `b10f94c` Kevin Chen style: Add 'eighth' to highighter.
- 2015-12-17 `5bad57c` Edward Li row boat demo
- 2015-12-17 `e88ff44` Edward Li added const key sigs
- 2015-12-17 `7f15bbd` Kevin Chen Add string concatenation.
- 2015-12-17 `a9e0b0e` Kevin Chen lib: Render: Fix long notes getting chopped up by short notes.
- 2015-12-17 `90a960b` Edward Li added tswift example
- 2015-12-17 `327581d` Kevin Chen examples_nh: twinkle: Fix syntax errors.
- 2015-12-17 `a1c1727` Kevin Chen lib: Octave, Relative, NormPitch, AddPitchNum, Parallel work now.
- 2015-12-15 `ae88764` Kevin Chen lib: Implement Render.
- 2015-12-17 `69b74f3` Kevin Chen scanner: Fix syntax error on lines with 'code //comment'.
- 2015-12-15 `c9aeb81` Kevin Chen test: array_basic tests arrays of typedefs now.
- 2015-12-17 `4d319d1` Edward Li added tempo to track concat
- 2015-12-17 `3215e5c` Kevin Chen lib: SameList: squash unused var warning.
- 2015-12-17 `49a9489` Kevin Chen test: Add cases for array concat.
- 2015-12-17 `3b927d8` Kevin Chen parser: Fix precedence of concat operator.
- 2015-12-16 `cda2831` Kevin Chen scanner: Reorganize rules into categories.
- 2015-12-16 `a060265` Kevin Chen scanner, parser: Rename ID tokens to ID_UPPER,LOWER.
- 2015-12-16 `9e18f4a` Kevin Chen lib: Size: Fix unused loop var.
- 2015-12-16 `9ea12f6` Kevin Chen lib: Extend: Fix output in padding-needed case.
- 2015-12-15 `c9522ee` Edward Li added demo twinkle test
- 2015-12-15 `5439afe` briank621 Added zip functionality and tests
- 2015-12-15 `7d125f9` Edward Li added parallel in std
- 2015-12-15 `786e4f3` Edward Li chords and octaves in lists now work
- 2015-12-15 `5fc116c` briank621 Added tests and std lib functions
- 2015-12-15 `e337a8c` Kevin Chen ast: Operator Eq should be printed as "=="
- 2015-12-15 `368177a` Kevin Chen lib: Fix ChordOfChords so it actually compiles.

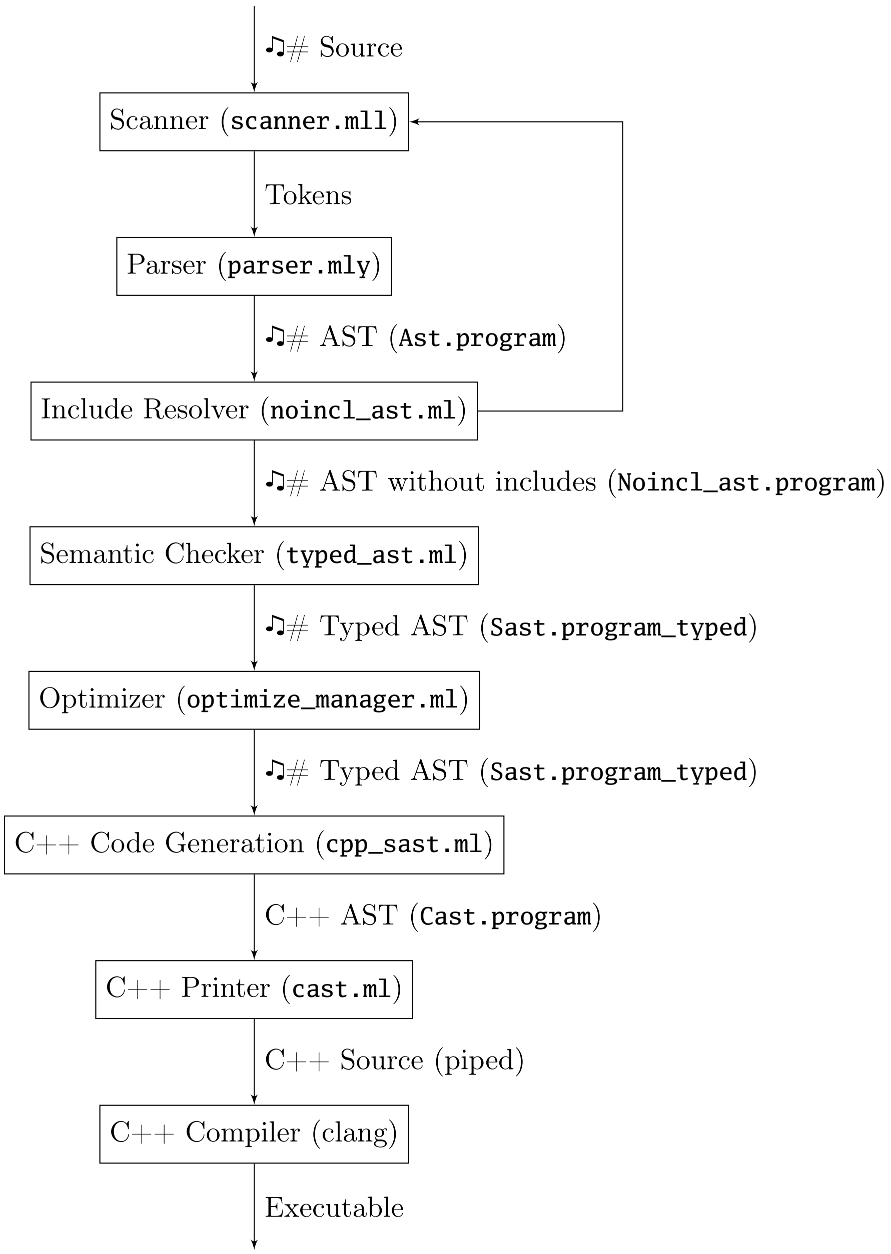
- 2015-12-15 `5644e69` Kevin Chen examples_nh: Fix twinkle.nh syntax.
- 2015-12-15 `98ef851` Kevin Chen lib: Move array, io, & math into separate files.
- 2015-12-15 `6df9ea6` Kevin Chen lib: Fix Range a b to return [a, b).
- 2015-12-15 `e371bdc` Kevin Chen noincl_ast: Better search, include actually works, refactor.
- 2015-12-15 `5d591f5` Kevin Chen style: Add nested comment highlighting.
- 2015-12-15 `cc96a76` Kevin Chen parser: octave @ no longer requires parens in array.
- 2015-12-12 `9e8d76a` briank621 Implemented operator== for typedefs
- 2015-12-13 `cf5b6f1` Kevin Chen examples_cpp: Move rendering to its own file.
- 2015-12-09 `ce2c165` briank621 Added range function and tests
- 2015-12-07 `3e9621e` Kevin Chen test: Improve coverage of array tests.
- 2015-12-07 `9534fb8` Kevin Chen parser: Move ArrIdx to non_apply.
- 2015-12-07 `3610276` Kevin Chen parser: C/Invoke for arrays and typedefs.
- 2015-12-07 `c3d2b41` Kevin Chen typed_ast: Clean up Arr/ArrMusic cases.
- 2015-12-07 `b959973` Kevin Chen test: Consistent naming for array tests.
- 2015-12-07 `e64af1d` Kevin Chen lib: support.hpp include guard.
- 2015-12-07 `19eae97` Kevin Chen typed_ast: Remove unused var tpitch in Uniop.
- 2015-12-07 `ce035ac` Kevin Chen parser: Less verbose BRACKS-counting code.
- 2015-12-06 `6226685` Edward Li moved volume to track
- 2015-12-06 `94b5fd5` Edward Li uncommented stdlib funs
- 2015-11-27 `566f69d` Kevin Chen [Fix #90] Implement for loops.
- 2015-12-06 `c47d3f0` Edward Li basic song object added
- 2015-12-06 `e2ba399` Edward Li uncommented more stuff in stdlib
- 2015-12-06 `80df6b4` Edward Li added music list test
- 2015-11-30 `59d85c9` Brian Kim [Fix #25, Fix #103] Empty lists and type check lists.
- 2015-12-06 `3a174d6` Edward Li got rid of manual construction of sast funapply in sastofast
- 2015-12-06 `c1a8839` Edward Li moved no fun ast validation to typed_ast
- 2015-12-06 `9173d00` Edward Li got rid of cast exit
- 2015-11-30 `001eb8c` Kevin Chen [Fix #32] Implement const (immutable values).
- 2015-12-02 `1648d15` Edward Li throw implemented
- 2015-12-02 `2cb305a` Edward Li functions now have access to globals
- 2015-12-02 `4f32344` Edward Li functions can now use user defined types (bugfix)
- 2015-12-01 `11feaab` Edward Li type inference for recursive functions
- 2015-12-01 `3db895b` Kevin Chen [Fix #51] Document git workflow.
- 2015-12-01 `9d498f9` Kevin Chen style: Highlight stdlib constants and unit.
- 2015-11-30 `b1ff737` Kevin Chen [Fix #62] ast, parser, sast: Use Core.Std.
- 2015-11-26 `6081a67` Kevin Chen [Fix #96] test: Add cases for binops.
- 2015-11-26 `1add618` Kevin Chen Fix undefined behavior in string literal comparison.
- 2015-11-26 `c974a93` Kevin Chen Guarantee the precedence of binops.
- 2015-11-24 `ce14712` Kevin Chen test: Add cases for binary operators.
- 2015-11-24 `84b0595` Kevin Chen [Fix #89] Type check and codegen for Conditional.
- 2015-11-24 `9640f9b` Kevin Chen typed_ast: Fix equality, comparison type-checking.
- 2015-11-24 `f1845f5` Kevin Chen std: Implement PrintEndline.
- 2015-11-29 `2ee61b1` Edward Li removed some unused value warnings from tests

- 2015-11-29 `be25e16` Edward Li fixed declaration retval, added tests
- 2015-11-26 `c890e6e` Kevin Chen [Fix #91] log: Flush output after each message.
- 2015-11-26 `710d8bf` Kevin Chen [Fix #95] parser: Fix error on single-line comments.
- 2015-11-24 `ae11021` Kevin Chen I found out why typedefs don't work
- 2015-11-24 `b9d6a9f` Kevin Chen [Fix #55] Improve parser & scanner error messages.
- 2015-11-23 `b4f622e` Kevin Chen tester: Don't ignore nhc or test program retvals.
- 2015-10-31 `c6c2521` Kevin Chen [Fix #26] parser: Implement tilde ˜for rest.
- 2015-11-23 `aecd484` Edward Li avoided using throw for now
- 2015-11-23 `bde8120` Edward Li fixed chord binop
- 2015-11-23 `505cd19` Edward Li uniops sast2cast
- 2015-11-23 `222c666` Edward Li binop sast2cast
- 2015-11-23 `2039510` Kevin Chen [Fix #74] nhc: Don't ignore Clang's return value.
- 2015-11-23 `407c823` Kevin Chen Turn on Clang address sanitizer.
- 2015-11-23 `857b216` Kevin Chen [Fix #83] Fix main return type errors.
- 2015-11-23 `8c6c1b5` Edward Li Fix fix block environment update.
- 2015-11-23 `88d5ab4` Edward Li binops ast2sast
- 2015-11-22 `7c7472f` Edward Li fixed block environment update
- 2015-11-22 `6c3aee6` Edward Li added signatures
- 2015-11-22 `6568c2f` Edward Li struct sast2cast cast2cpp
- 2015-11-22 `31b7c5c` Edward Li fixed dependency checking bug
- 2015-11-22 `9603631` Edward Li init and assign sast2cast
- 2015-11-22 `6475246` Edward Li checked for mutually recursive types
- 2015-11-22 `26dc4a9` Edward Li structs ast2sast
- 2015-11-22 `625037f` Edward Li assignment and init ast2sast
- 2015-11-21 `b64897d` Edward Li set up sast2cast for all possbile sast
- 2015-11-20 `f7493c2` Edward Li var ref ast2sast
- 2015-11-19 `2e87f2f` Kevin Chen support: Add concatenation helper.
- 2015-11-18 `5698009` Kevin Chen [Fix #78] Create C++ AST based on Micro C compiler.
- 2015-11-18 `78a472f` Stephen Edwards Add C++ AST based on Micro C compiler.
- 2015-11-16 `48f5856` Kevin Chen typed_ast: Improve error when function typecheck fails.
- 2015-11-16 `6ff7446` Kevin Chen parser: Fix grammar so fun_def doesn't use ==.
- 2015-11-16 `0f9ceda` Kevin Chen nhc: Turn on clang -Weverything.
- 2015-11-16 `65e427f` Kevin Chen typed_ast: Add support for Block.
- 2015-11-16 `1941f91` Kevin Chen nhc: Make keep_il option actually work.
- 2015-11-16 `616792d` Kevin Chen lib: Add int/float/string conversion functions.
- 2015-11-16 `85a387a` Kevin Chen Fix duplicated .nh.nh when reading toplevel.
- 2015-11-15 `ccce050` Kevin Chen tester: Make the implementation more functional.
- 2015-11-16 `8e03b48` Edward Li renamed h to hpp
- 2015-11-15 `3fb4de6` Edward Li nhc pipeline
- 2015-11-15 `8ac76c7` Kevin Chen [Fix #57] nhc: Uncaught failwiths are now logged.
- 2015-11-15 `75ac60c` Edward Li recursively add includes
- 2015-11-15 `fa3a5a6` Edward Li renamed functions, wrapped outer in main
- 2015-11-15 `bbae2b3` Edward Li basic cpp gen sufficient for hello world
- 2015-11-15 `7b29203` Kevin Chen tester: Fix indentation.

- 2015-11-15 `643de17` Kevin Chen tester: Integrate with makefile.
- 2015-11-15 `9285c7d` Kevin Chen [Fix #65] nhc: Improve descriptions in help message.
- 2015-11-15 `f697206` Kevin Chen ast: Put newlines between functions printed.
- 2015-11-09 `2b3d340` Kevin Chen [Fix #29] parser: Add syntax for calling C++ functions.
- 2015-11-09 `845bc01` Kevin Chen scanner: Prioritize keywords above identifiers.
- 2015-11-15 `58d81bd` Edward Li made a line simpler
- 2015-11-15 `93e3349` Edward Li top level program spacing now fixed
- 2015-11-15 `7cd921e` briank621 Fixed the while loop
- 2015-11-15 `b8f1ec9` briank621 Changed function to really_input_string
- 2015-11-15 `9ff47be` briank621 Tester functionality works for test cases
- 2015-11-13 `0baf40e` Edward Li implemented literals and funApp in sast
- 2015-11-09 `ef595f2` Kevin Chen nhc: No need to open files as binary.
- 2015-11-09 `fec9453` briank621 Added code to compile and test the output string of .nh files
- 2015-11-09 `9322c5e` Kevin Chen Makefile: Fix version.ml error message on clean.
- 2015-11-09 `b143d67` briank621 tester prints out directory structure. Added hello world test cases
- 2015-11-09 `fff7172` Kevin Chen nhc: Fix unused variables keep_il and keep_ast.
- 2015-11-09 `f0d5072` Kevin Chen Move AST dumper to ast.ml.
- 2015-11-09 `c4c98cf` Kevin Chen Move .nh files out of src.
- 2015-11-06 `07b87c3` Jennifer Lam [Fix #13 #35] Added assignments to AST and assignment lists to parser
- 2015-11-06 `021517c` briank621 Added ast dumping functionality for throw, chord, zip, octave
- 2015-11-02 `48c8882` briank621 Added functionality for chords, zip, octave and unary sharp/flats
- 2015-11-03 `eaa42a9` Kevin Chen nhc: -A option actually dumps AST now.
- 2015-11-03 `6e04b17` Kevin Chen Update build info automatically based on git.
- 2015-11-02 `23845c6` Kevin Chen [Fix #44] nhc: Parse command-line options according to LRM.
- 2015-11-01 `0f1c04b` Kevin Chen [Fix #45] log: Create logging module.
- 2015-11-02 `a0f4562` Edward Li throw keyword
- 2015-10-23 `e24e3a1` Jennifer Lam Fixed bug: allows quotations to be in the middle of the string without the string terminating.
- 2015-11-02 `5add02d` Edward Li fixed function call bug
- 2015-11-02 `12f20a9` Edward Li added some example standard library files
- 2015-11-02 `ac0179c` Edward Li added sample nh programs
- 2015-11-01 `1ab43a1` Kevin Chen test.ml: camlp4 reformat.
- 2015-11-01 `61c7677` briank621 Updated test.ml to reflect the program structure
- 2015-11-01 `3247e3d` Edward Li modified twinkle for various instruments
- 2015-11-01 `b60d9ad` Kevin Chen parser: Allow no-parameter function calls.
- 2015-11-01 `410d84a` Kevin Chen ast: Change fundef to tuple instead of record.
- 2015-10-31 `e9b0494` Kevin Chen parser: Includes no longer required to be at end.
- 2015-10-30 `f86ee13` Kevin Chen [Fix #24] parser: Implement includes.
- 2015-10-30 `228630e` Kevin Chen [Fix #20] parser: < <= > >= associativity & priority.
- 2015-10-30 `03552f6` Kevin Chen parser: Function application not required to have parens.

- 2015-10-30 `1b17238` Kevin Chen ast: Change fundef to use bytes instead of string.
- 2015-10-26 `1803054` Edward Li fixed For loop bug
- 2015-10-24 `ccd46ce` briank621 ast dumper works for literal 123, but not for the sample program
- 2015-10-24 `8eb8a69` briank621 Adding updates to ast_dumper
- 2015-10-23 `500f977` Kevin Chen [Fix #10] Implement function definition.
- 2015-10-23 `b80601f` Kevin Chen Add parser intermediate files to .gitignore.
- 2015-10-23 `1cdf4bc` Edward Li implemented control flow
- 2015-10-21 `486e3d9` Kevin Chen [Fix #7] Implement float literals in scanner.
- 2015-10-21 `81455a2` Kevin Chen [Fix #11] Implement comments in scanner.
- 2015-10-22 `021862c` Edward Li made line continuation no longer gobble ';' and added crlf as a possible SEP
- 2015-10-22 `5862c71` Edward Li implemented scoping, line continuations, and multiple lines; also changed entry point from expr to block
- 2015-10-15 `156d414` Kevin Chen Implement grammar for boolean logic.
- 2015-10-15 `2fb60b2` Kevin Chen Implement grammar for concatenation.
- 2015-10-15 `5aa6727` Kevin Chen Add grammar support for boolean comparisons.
- 2015-10-15 `4da2829` Kevin Chen Implement negative numbers.
- 2015-10-15 `75d16a0` Kevin Chen Only allow application on ID_FUN.
- 2015-10-15 `c8c2b17` Kevin Chen [Fix #2] Reorganize parser rules & implement func application.
- 2015-10-15 `25889bb` Kevin Chen install_dependencies.sh now gets OCaml too.
- 2015-10-15 `4f3fd0b` Kevin Chen Switch compilation to corebuild.
- 2015-10-14 `8d6cc12` Kevin Chen Fix list parsing. Fix separator parsing.
- 2015-10-14 `339e3ed` Kevin Chen Add grammar with binops, literals, and identifiers.
- 2015-10-14 `23eefec` Kevin Chen Fix stk submodule location.
- 2015-10-14 `07dfbcf` Jennifer Lam installing stk dependencies
- 2015-10-14 `8a05c25` Jennifer Lam added git submodue for stk library.
- 2015-10-14 `34c81dd` Edward Li fixed dependecy file bug
- 2015-10-14 `a47b173` Jennifer Lam Fixed makefile
- 2015-10-14 `c4c941a` Edward Li added clang to dependency script
- 2015-10-14 `96be91b` Edward Li updated dependency script
- 2015-10-07 `7c131a4` Kevin Chen Add C++ example of Twinkle.
- 2015-10-06 `2f99deb` Kevin Chen Initial commit.

5 Architecture (Kevin)


♫# Source
Scanner (scanner.mll)
Tokens
Parser (parser.mly)
♫# AST (Ast.program)
Include Resolver (noincl_ast.ml)
♫# AST without includes (Noincl_ast.program)
Semantic Checker (typed_ast.ml)
♫# Typed AST (Sast.program_typed)
Optimizer (optimize_manager.ml)
♫# Typed AST (Sast.program_typed)
C++ Code Generation (cpp_sast.ml)
C++ AST (Cast.program)
C++ Printer (cast.ml)
C++ Source (piped)
C++ Compiler (clang)
Executable

5.1 Toplevel (`nhc.ml`)

The `Nhc` module contains the entry point to our compiler. It generates an argument parser using Core's Command library, then calls each stage of the compiler on the program provided. If any stage encounters an unrecoverable error, it throws an exception. The toplevel catches the exceptions, prints their messages, and exits the program with a nonzero exit code.

Depending on the arguments given by the user, the toplevel may also dump the abstract syntax tree and C++ source code.

5.2 Include Resolver (`noincl_ast.ml`)

The include resolver recursively calls the scanner and parser on files specified by `include` statements. It also ensures that files are not included twice, so include loops are traversed only once.

5.3 Scanner (`scanner.mll`)

The scanner generates tokens, which are keywords, identifiers, operators, literals, and symbols. Apart from matching regular expressions, its tasks are:

- Converting escape sequences in string literals.
- Removing comments and ensuring that multiline comments are properly nested (each `/*` has a matching `*/`).
- Removing whitespace, except for newlines, which are passed onto the parser as tokens.

The scanner does not resolve `include` statements. Doing so would require it to know the header search paths from the command-line arguments, which is beyond the scope of the scanner's responsibilities.

5.4 Parser (`parser.mly`)

The parser produces an abstract syntax tree from the token stream. It uses whitespace tokens (`SEP`) to tell when a line ends

It is sometimes valid to have extra `SEP` tokens (between lines of code) or none at all (after the `{` in `type t = { ... }`). We defined these helper productions to eat up newlines:

```
/* Helper: One or more separators */
sep_plus:
| SEP { () }
| SEP sep_plus { () }

/* Helper: Zero or more separators */
sep_star:
| /* nothing */ { () }
| SEP sep_star { () }
```

5.5 Semantic Checker (`typed_ast.ml`)

The semantic checker enforces the rules described in the Reference Manual, including type safety and mutability. If the AST complies with these rules, the semantic checker emits a semantically checked AST (SAST).

This is the last stage where an error can be emitted. If a program passes the semantic checker, all future stages will succeed — we are not using the C++ compiler as a crutch to enforce, for example, `const`.

5.5.1 Type Inference

Our language uses an ad-hoc type inference algorithm. The type of an expression is built bottom-up from literals and variables, as their types are already known. Functions use implicit compile-time duck typing similar to C++ templates. We check their types during function application by recursively calling the semantic checker on the body of the function using type information from the arguments.

Finding the return value of a recursive function is a little more involved. This is done in two passes. First, we assume the recursive branch has the same type as the base case's, and store the function's inferred return type for the given arguments. (Infinite recursion is not allowed.[18]) In the second pass, we make sure our assumption in the first part was correct.

A consequence of our primitive type inference algorithm is that the programmer must specify the type of empty array literals: for example, `int{}`. However, not having to learn and implement a "real" type inference algorithm helped us develop the language more quickly.

5.5.2 User-defined Types

User-defined types are allowed to contain members that are other user-defined types, as long as they are not mutually recursive. The semantic checker builds a graph of these relationships, and ensures there are no cycles.

5.6 Optimizer (`optimize_manager.ml`, `optimize.ml`)

The optimizer consists of a list of functions from `Optimize`, which `Optimize_manager` calls on the SAST in order.[19]

The only optimization in our compiler is constant folding (`Optimize.constfold`). It runs a depth-first search on the SAST, finding binary operator nodes where both sides are literals. Nodes matching this pattern are passed to interpreter (`interpret.ml`) — the result returned from the interpreter is used to replace the node in the SAST.

5.7 C++ Code Generation (`cpp_sast.ml`)

5.7.1 Ensuring memory safety

Our language is supposed to guarantee memory safety: no matter what the programmer writes, their programs should not be able to leak memory. C++'s zero-cost abstractions allow us to achieve these goals without the overhead of a garbage collector.

In our generated C++ code, both user-defined types and arrays (implemented with C++ vectors) are stored on the stack. When an object leaves scope, its destructor is automatically called. For user-defined types, C++ implicitly generates a destructor. The vector class has a destructor that calls the destructor of each element of the vector.

The move constructor ensures that a vector's contents do not have to be copied when it is returned from a function.

[18] C++ has a similar restriction: it cannot compile a function template where the return value is also templated.
[19] Our compiler design philosophy can be summarized as "when in doubt, copy LLVM."

5.7.2 unit_t provides an ML-style unit

To pass and store unit `()` values in C++, we defined a new type `unit_t`:

```
typedef int unit_t;
const unit_t LIT_UNIT = 0;
```

A function that returns `LIT_UNIT` has type `unit_t`. Two values of type `unit_t` will always have the value 0, equality works as expected.

5.7.3 Functional programming in an imperative language

In our language, everything has a return value — there's no distinction between expressions and statements. For example, we can return a value from blocks and conditionals. Even the unit returned by an assignment can be assigned:

```
y = (x = (Ignore 1; Ignore 2; 3)) // => x = 3, y = ()
greeting = if audience < 7*1000*1000*1000 then "Hello World" else "Hello Universe"
```

However, the same is not true of C++. To get around this, we wrap everything in lambdas:

```
// Line 1
unit_t y = [&] () -> unit_t {
  int64_t x = [&] () -> int64_t { // Create a lambda from the block
    Ignore(static_cast<int64_t>(1));
    Ignore(static_cast<int64_t>(2));
    return 3; // "Block" returns its last expression
  } (); // Call the lambda, and assign the result to x
  return LIT_UNIT; // Assignment returns unit, which is assigned to y
} ();

// Line 2
[&] () -> unit_t {
  std::string greeting = [&] () -> std::string {
    if ((audience) < (7000000000)) // Evaluate the conditional
      return [&] () -> std::string {
        return std::string("Hello World");
      } (); // If true, call and return the first lambda
    else
      return [&] () -> std::string {
        return std::string("Hello Universe");
      } ();
  } (); // Call the conditional's lambda, and assign the result to greeting
  return LIT_UNIT;
} ();
```

Although our abuse of lambdas slows down C++ compilation, it is easy to see that this code generation algorithm is correct over all input programs. It's also easy to implement.

5.8 C++ Compiler (clang)

We chose clang because it has AddressSanitizer, which instruments programs at compile-time to catch common memory errors, such as use-after-free and out-of-bounds access. This was extremely useful for hunting down code generation bugs.

5.9 Compiler Roles

Because of our development strategy, all members played a role in implementing most components. The table below is based on Git blame.

Component	**Authors**
`Makefile`	Kevin
`ast.ml`	Edward, Kevin, Brian
`cast.ml`	Kevin, Edward
`cpp_sast.ml`	Edward
`interpret.ml`	Kevin
`log.ml`	Kevin
`nhc.ml`	Kevin
`noincl_ast.ml`	Edward, Kevin
`optimize.ml`	Kevin
`parser.mly`	Kevin, Edward
`sast.ml`	Edward
`scanner.mll`	Kevin
`support.cpp`	Kevin
`tester.ml`	Kevin, Brian
`typed_ast.ml`	Edward, Kevin
Standard library	Brian, Edward, Kevin

6 Test Plan (Brian)

6.1 Factorial Example

6.1.1 Source Code in native language

The following code implements the `Factorial` function recursively using conditionals:

```
fun Factorial x = if x <= 1 then 1 else x * Factorial (x-1)
x = Factorial 3
Print (StringOfInt x)
```

6.1.2 Source Code in target language

The Factorial is implemented with an `if` statement, recursively calling itself if the argument is greater than 1, otherwise returning 1. The generated C++ source code is shown below:

```
#include <iostream>
#include <string>
#include <vector>
#include "support.hpp"

int64_t Factorial(int64_t);

int main(){
return [&] () -> int64_t {
return [&] () -> int64_t {
[&] () -> unit_t {
[&] () -> unit_t {
  //call Factorial
x = Factorial(static_cast<int64_t>(3));
return LIT_UNIT; } ();
  //print result
return nh_support::print_string(
static_cast<std::string>([&] () -> std::string {
return std::to_string(static_cast<int64_t>(x));
} ())); }
();return 0; }
();}();}

int64_t Factorial(int64_t x){
return [&] () -> int64_t {
return [&] () -> int64_t {
  //base case
if ((x) <= (1))
return [&] () -> int64_t {
return 1;}();
else
  //recursive case
return [&] () -> int64_t {
return (x) * (Factorial(static_cast<int64_t>([&] () -> int64_t {
return (x) - (1);}())));}();}();}();}
```

6.2 Musical Scale Example

6.2.1 Source Code in native language

The following code demonstrates the use of the standard library `Scale` function as well as the `AddPitchNum` and `NormalizePitch` functions that are subsequently called.

```
fun Scale pitch_start pitch_end = (
  c = chord{}
  oct_diff = pitch_end$octave - pitch_start$octave
  pitch_diff = pitch_end$rank - pitch_start$rank
  total_diff = (Size key_sig$scale) * oct_diff + pitch_diff
  direction = be 1 unless total_diff < 0 inwhichcase -1
  for x in Range 0 (direction * total_diff + 1) do
    c = c . [(AddPitchNum (direction*x) pitch_start key_sig)]
  c
)

fun AddPitchNum num p keysig = (
  p$rank = p$rank + num
  NormalizePitch p keysig
)

fun NormalizePitch p keysig = (
  scale_size = Size keysig$scale
  if p$rank < 1 then (
    octave_offset = (-1 * p$rank) / scale_size + 1
    p$rank = p$rank + octave_offset * scale_size
    p$octave = p$octave - octave_offset
  ) else (
    p$rank = p$rank - 1 // Pitch is [1, 7] -> [0, 6]
    p$octave = p$octave + p$rank / scale_size
    p$rank = p$rank % scale_size
    p$rank = p$rank + 1 // Undo our change: [0, 6] -> [1, 7]
  )
  p
)

(
large_scale = Scale (5@(-1)) (3@1)
manual_scale = [ 5@(-1) 6@(-1) 7@(-1) 1 2 3 4 5 6 7 1@1 2@1 3@1 ]

if large_scale == manual_scale then Print "Great" else Print "Not Great"
)
```

6.2.2 Source Code in target language

This program tests the functionality of scale. We declare `large_scale` by calling the Scale function with arguments $(5@(-1))$ and $(3@-1)$. The Scale function should return all the notes in between these two, as shown by the elements in `manual_scale`. If `large_scale` and `manual_scale` are equal, the program will print `Great`.

Here is the generated C++ source code:

```
#include <iostream>
#include <string>
#include <vector>
#include "support.hpp"
//STRUCT DECLARATIONS AND DEFINITIONS TRIMMED

//CONST VARIABLES AND EQUALITY FUNCTIONS TRIMMED

int main()
{
//NOTE, SCALE, KEY AND TIME SIGNATURE INITIALIZATION TRIMMED
return [&] () -> int64_t {
return [&] () -> int64_t {
[&] () -> unit_t {

  //instantiate large_scale
return [&] () -> unit_t {
std::vector<struct chord> large_scale = Scale(static_cast<struct
pitch>([&] () -> pitch {
return AddPitchOctave(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(5))),
       static_cast<int64_t>([&] () -> int64_t {
return -1; }
()));}
()), static_cast<struct pitch>([&] () -> pitch {
return AddPitchOctave(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(3))),
static_cast<int64_t>(1)); } ()));

  //instantiate manual_scale
std::vector<struct chord> manual_scale =
std::vector<struct chord>({
  ChordOfPitch(static_cast<struct pitch>(AddPitchOctave(
  static_cast<struct pitch>(PitchOfInt(static_cast<int64_t>(5))),
  static_cast<int64_t>([&] () -> int64_t {
```

```
return -1; }
()))), ChordOfPitch(static_cast<struct
pitch>(AddPitchOctave(static_cast<struct pitch>(
PitchOfInt(static_cast<int64_t>(6))), static_cast<int64_t>([&] () ->
int64_t { return -1; }
()))), ChordOfPitch(static_cast<struct pitch>
(AddPitchOctave(static_cast<struct pitch>(
PitchOfInt(static_cast<int64_t>(7))), static_cast<int64_t>([&] ()
-> int64_t { return -1; }
()))), ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(1)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(2)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(3)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(4)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(5)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(6)))),
ChordOfPitch(static_cast<struct
pitch>(PitchOfInt(static_cast<int64_t>(7)))),
ChordOfPitch(static_cast<struct pitch>
(AddPitchOctave(static_cast<struct pitch>
(PitchOfInt(static_cast<int64_t>(1))), static_cast<int64_t>(1)))),
ChordOfPitch(static_cast<struct pitch>
(AddPitchOctave(static_cast<struct pitch>
(PitchOfInt(static_cast<int64_t>(2))), static_cast<int64_t>(1)))),
ChordOfPitch(static_cast<struct pitch>
(AddPitchOctave(static_cast<struct pitch>
(PitchOfInt(static_cast<int64_t>(3))), static_cast<int64_t>(1))))});
return [&] () -> unit_t {

//check for equality and print corresponding message
if ((large_scale) == (manual_scale))
return [&] () -> unit_t {
return nh_support::print_string(static_cast<std::string>(
std::string("Great")));}();
else
return [&] () -> unit_t {
return nh_support::print_string(static_cast<std::string>(
std::string("Not Great")));
}();}();}();}();return 0;}();}();}

std::vector<struct chord> Scale(pitch pitch_start, pitch pitch_end){
return [&] () -> std::vector<struct chord> {
return [&] () -> std::vector<struct chord> {
```

```
std::vector<struct chord> c = std::vector<struct chord>({  });
int64_t oct_diff = (pitch_end.octave) - (pitch_start.octave);
int64_t pitch_diff = (pitch_end.rank) - (pitch_start.rank);
int64_t total_diff = (([&] () -> int64_t {
return Size(static_cast<std::vector<double>>(key_sig.scale));}
()) * (oct_diff)) + (pitch_diff);
int64_t direction = [&] () -> int64_t {
if ((total_diff) < (0))
return [&] () -> int64_t { return -1;}();
else
return [&] () -> int64_t { return 1;} ();} ();
[&] () -> unit_t {
for (int64_t x :
Range(static_cast<int64_t>(0), static_cast<int64_t>([&] () ->
int64_t { return ((direction) * (total_diff)) + (1);}
()))) [&] () -> unit_t {
c = nh_support::concat(c, std::vector<struct chord>({
ChordOfPitch(static_cast<struct pitch>([&] () -> pitch {
return AddPitchNum(static_cast<int64_t>([&] () -> int64_t {
return (direction) * (x);}
()), static_cast<struct pitch>(pitch_start),
static_cast<struct key_signature>(key_sig));} ())) }));
return LIT_UNIT;}
(); return LIT_UNIT;}
(); return c;} ();} ();}

pitch AddPitchNum(int64_t num, pitch p, key_signature keysig) {
return [&] () -> pitch {
return [&] () -> pitch {
[&] () -> unit_t {
p.rank = (p.rank) + (num);
return LIT_UNIT; } ();
return NormalizePitch(static_cast<struct pitch>(p),
static_cast<struct key_signature>(keysig)); } ();} ();}

pitch NormalizePitch(pitch p, key_signature keysig){
return [&] () -> pitch {
return [&] () -> pitch {
int64_t scale_size =
Size(static_cast<std::vector<double>>(keysig.scale));
[&] () -> unit_t {
if ((p.rank) < (1))
return [&] () -> unit_t {
int64_t octave_offset = (([&] () -> int64_t {
return (-1) * (p.rank);}
()) / (scale_size)) + (1);
[&] () -> unit_t {
p.rank = (p.rank) + ((octave_offset) * (scale_size));
return LIT_UNIT; } ();
return [&] () -> unit_t {
p.octave = (p.octave) - (octave_offset);
return LIT_UNIT; } (); } ();
else
return [&] () -> unit_t {
[&] () -> unit_t {
p.rank = (p.rank) - (1);
return LIT_UNIT; } ();
[&] () -> unit_t {
p.octave = (p.octave) + ((p.rank) / (scale_size));
return LIT_UNIT; } ();
[&] () -> unit_t {
p.rank = (p.rank) % (scale_size);
return LIT_UNIT; } ();
return [&] () -> unit_t {
p.rank = (p.rank) + (1);
return LIT_UNIT; } (); } (); }
(); return p; } (); } (); }

//Size, Range, AddPitchOctave, ChordOfPitch
//PitchofInt, PrintEndline, Ignore Trimmed
```

6.3 Test Automation

Our testing automation program (invoked by calling `make test`) iterates through each file in the directory ending in `.nh`. After running the program, the tester compares the results from stdout with the corresponding `.out` file and prints a line to the console with the result.[20]

```
$ make test
./update_version.sh
corebuild -use-ocamlfind -cflags -safe-string nhc.native
Finished, 52 targets (0 cached) in 00:00:09.
corebuild -use-ocamlfind -cflags -safe-string tester.native
Finished, 8 targets (4 cached) in 00:00:01.
./tester.native ../test
debug: ✅ array_basic.nh
debug: 🔥 array_empty.nh
debug: ✅ array_equality.nh
debug: ✅ array_music_chords.nh
debug: ✅ array_music_float.nh
...
```

6.4 Test Cases

The `test` directory contains our test cases. As of commit `b57683f`, there are 73 tests. For any feature to be pushed into our language, we required at least one test demonstrating its

[20]Printing emoji to the console is an essential feature of our tester.

functionality ensuring seamless integration. We tested the following features of our language:

Assignment. We tested basic declarations and assignments. Furthermore, we tested assignment to `const` variables as well as nested assignments: `x = (y = "what" )`, to check for the unit return type. (`assign_*.nh`, `simple_assign.nh`, `simple_varref.nh`)

Operators. We tested operations with the data types, from integers, booleans, and arrays to musical types such as pitches. Operations also included boolean and comparison operators. (`ops_*.nh`, `flatsharp.nh`)

Printing. We tested print functionality for both strings and non-string types, required for our "Hello World" demo. (`print_*.nh`)

Comments. We tested our comment functionality, including single-line and multi-line comments. (`comment_*.nh`)

Control Flow. We tested conditional statements (if, be-unless-inwhichcase), for loops and the throw statement. Nested if statements and for loops, including those with multi-line bodies, were included. (`if_*.nh`, `lit_bool.nh`, `for_*.nh`, `throw_*.nh`)

Arrays and Music Arrays. The building block for the songs in our program is chords, which are represented through arrays. Thus extensive testing was done for arrays, checking musical arrays (single notes, notes with octaves, simultaneous notes), standard arrays, nested arrays and empty arrays. (`array_*.nh`)

Block Structure. Blocks are enclosed via parentheses so we tested this functionality with variable assignments. (`simple_block.nh`)

Function Definition and Application. We tested declaring both recursive and non-recursive functions. We also tested function overloading, an important feature of our language, and passed in different types to the same function. (`fundef_*.nh`, `globals_fundef.nh`)

User-defined Types. We tested declaring, instantiating, referencing and accessing fields for user defined types. We also tested default value functionality and more complicated types, such as nested types. (`typedef_*.nh`, `nested_types.nh`)

Standard Library. We tested each function defined in the standard library. Most of the standard library functions work with the musical types, such as tracks and chords. Included in these tests is an implementation of Twinkle Twinkle in our language. (`std_*.nh`, `demo_twinkle.nh`)

6.5 Testing Roles

The testing infrastructure was created by Brian and Kevin. Tests were written by the person who implemented the feature. All team members were responsible for filing tickets on bugs they encountered. For more information on feature responsibilities, see the Project Log in section 4.7.

7 Lessons Learned

7.1 Kevin Chen

Automation is worth it. Over the semester, automated testing helped me catch about a dozen regressions I'd introduced. Our automated tests were useful because they were fast, easy to write, easy to run (`make test`), and reliable (no flaky tests). Without these properties, we wouldn't have used the tests and development would've moved more slowly. In general, if something is boring but super important, automating it will pay huge dividends down the road.

Code review: less is better. We caught a lot of bugs and oversights through code review. But we were too conservative about merging in smaller (30 lines or fewer) changes. It would've been more efficient to merge these smaller changes optimistically as long as the tests passed — the benefits of unblocking tasks outweigh the cost of maybe having to refactor the code later.

Functional programming improves software quality. I learned that functional languages' restrictions are a good thing: they enable correctness guarantees that are impossible in imperative languages. The OCaml compiler gave us the confidence to take on large refactors knowing that we wouldn't cause strange runtime errors. OCaml also forced me to think more about what I was trying to accomplish, making the algorithm more elegant in the process.

Visibility speeds up development. Our original semantic checker implementation gave the same "couldn't find function" message for three different failure modes: the function didn't exist, call had the wrong number of arguments, and wrong types. It also didn't tell which function call it was complaining about. By printing separate, detailed diagnostic messages for each case, we made it easier to debug semantic checking edge cases for functions. When working on a complex project, we needed as much visibility as we could get.

7.2 Brian Kim

This project is likely the biggest one I have worked on. Working on a product that involves a substantial amount of code required a neatness that I didn't bring to my code-writing before. Clean and compact code was emphasized in the group. Furthermore as this wasn't an individual project, we needed certain coding standards and good version control. I learned a lot about git, particularly in re-basing code and code review. The practices in this group project carry over well into developing products in general. I'm glad that we worked relatively early and consistently, bypassing the need for a mad rush at the end of the semester. Besides working in a group, I learned the intricacies of the compiler and ended up building something I never thought I would have. I learned the convenience of representing a program abstractly through syntax trees. I added a new language to my arsenal, and programming in a functional language really helped sanitize my mind from the habits of an imperative programming background.

7.3 Edward Li

Well the lesson that is immediately apparent here is that don't be the last one to contribute to a section of the report because you'll be left with the scrapings. Automated testing, code reviews, OCaml's type system — as with my teammates, I have discovered the immense utility of each of these things, and I would recommend every group take advantage of them. The theme here is that for any project of nontrivial size, maintainability is key. You will invariably have to go back and change some parts of the code you've already written; being able to make those changes with the confidence that you are making the right changes and are not breaking anything else allows your group to move at exponential pace. In summary, you absolutely need correctness and speed - and you get those with automated testing, code reviews, and functional OCaml.

One of the key factors to our successs in implementing so many features is that we were constantly thinking ahead. We were always thinking about the dependencies of the next features we were going to implement. This helped us prioritize and distribute the work so that no members would be waiting on someone else to complete their feature. It is also important to think outside of just the language features. For example, as we got closer to having a complete language, the next step was the standard library. This is a new language that nobody has ever written in, so to ensure that productivity did not drop, I added a syntax highlighter for Sublime Text. The result is that we were able to write hundreds of lines in our standard library, completely in a new language that we had just designed. This is just one example of the many things we did outside of just implementing the compiler. Other examples include: setting up automated testing, playing around with the target language before actually trying to translate into it, writing plenty of examples in your language early on to make sure it's actually usable and consistent, making sure every member is using the same development environment, writing setup/install scripts so everyone has the same version of dependencies, and more. In summary, think ahead, remove any potential blocks to productivity, and prioritize savagely for both implementing the language and the features outside of the language itself.

7.4 Advice for Future Teams

- Use GitHub! It has many tools to document not only your progress, but also what features have yet to be implemented. This was key to our planning and scheduling. It really helped our team keep a healthy pace by visualizing exactly what issues were left to resolve/implement. Plus, it's super satisfying to close a ticket because you get to click the "close issue" button.
- Write the parser concurrently with the LRM. This will help you weed out any major issues early on.
- Read ahead in the slides so you can start your project early. Following Prof. Edwards' schedule runs the risk of not finishing the compiler on time. It will also help you avoid any potential complications. For example, if we had known how difficult throw/catch would be to implement, we probably would have included an "any type" in our language.
- Absolutely make the effort to write in the functional style — it will help guarantee correctness and elegance.
- Learn about OCaml's mutability, but only use it when you have no other options.
- Meet regularly (1–2 times a week) even if it seems like there's nothing to do. This project is a marathon, not a sprint.
- Make sure everyone is always on the same page — for both understanding the state of the project and the pace needed to complete all the milestones. Make use of frequent meetings to guarantee this.
- Keep people excited and interested in the project. This will help productivity.

8 Future Plans

If we had more time to work on the compiler, we would implement:

- Add line numbers to compiler errors, making the language more usable. Perhaps use a different parser generator, or write our own recursive-descent parser like clang's.
- More robust type inference algorithm. No more typing `string{}`! And we can finally fix exceptions.
- REPL. This will allow users to experiment with languages interactively.
- Higher quality synthesis, including the ability to use multiple instruments. (This is a standard library change, not a language change.)
- Unused and unassigned variable warnings. These will help users write cleaner code and encourage the use of `const`.
- Name mangling. Currently, it's not possible to create a variable named `continue` because that's a keyword in C++.

9 Appendix

9.1 Sample Programs

Our examples are in the `examples_nh` folder in Code Listings, section 9.2. Examples include:

- Pachelbel's Canon
- "Twinkle, Twinkle Little Star"
- "Row, Row, Row Your Boat"
- "You Belong With Me" by Taylor Swift
- "Stairway To Heaven" by Led Zeppelin
- "Walk This Way" by Aerosmith
- Sample music on an Arabic scale, which has pitches that are not found on the Western scale

These examples demonstrate the ease of use and the structure of the music. Our language is powerful enough to provide the same abstractions to any style of music you wish to compose, even if you are using frequencies no one has ever heard before.[21]

[21] Literally. You could create dog music by defining a key signature with frequencies higher than can be captured by the human ear, but still within that of dogs' — though you may need to upgrade your audio hardware to play it.

9.2 Code Listings

9.2.1 .gitignore

```
#OCaml
_log
_build
*.cmi
*.cmo
*.native
*.byte
parser.output
parser.ml
parser.mli

#nhc
/src/version.ml

#C++
*.o
core
a.out

#Audio files
*.wav
*.mp3
*.midi
*.mid

#Mac
.DS_Store
._*

#Windows
Thumbs.db
```

9.2.2 .gitmodules

```
[submodule "stk"]
        path = stk
        url = https://github.com/thestk/stk
```

9.2.3 examples_cpp/Makefile

```
CC  = clang
CXX = clang++

INCLUDES = -I/usr/local/include/

FLAGS    = -Wall -pedantic -O2 $(INCLUDES)
CFLAGS   = $(FLAGS)
CXXFLAGS = $(FLAGS) -std=c++1z

LDFLAGS = -L/usr/local/lib/
LDLIBS  = -lstdc++ -lstk -lm

.PHONY: default
default: twinkle

twinkle: twinkle.o music_utils.o

music_utils.o: music_utils.cpp music_utils.hpp

.PHONY: clean
clean:
        rm -f *~ *.wav *.o core twinkle

.PHONY: all
all: clean default
```

9.2.4 examples_cpp/music_utils.cpp

```
#include "music_utils.hpp"

float to_frequency(int note)
{
    note -= 10;
    return 440.0 * pow(2.0, (double)note / 12.0);
}

std::unordered_map<int, double> scale
{
    { 1, to_frequency(1) },
    { 2, to_frequency(3) },
    { 3, to_frequency(5) },
    { 4, to_frequency(6) },
    { 5, to_frequency(8) },
    { 6, to_frequency(10) },
    { 7, to_frequency(12) },
};
```

9.2.5 examples_cpp/music_utils.hpp

```
#ifndef __music_utils_hpp__
#define __music_utils_hpp__

#include <iostream>
#include <unordered_map>

#include <stk/Instrmnt.h>
#include <stk/FileLoop.h>
#include <stk/FileWvOut.h>

// C4 when note == 1
float to_frequency(int note);

template <typename TSong, typename TInstrument>
void render(std::string filename, const TSong &song, TInstrument instrument, const int note_length = 15000);

extern std::unordered_map<int, double> scale;

// Template implementations
template <typename TSong, typename TInstrument>
void render(std::string filename, const TSong &song, TInstrument instrument, const int note_length)
```

```cpp
{
    // Set the global sample rate before creating class instances.
    stk::Stk::setSampleRate(44100.0);
    stk::Stk::setRawwavePath("../stk/rawwaves/");

    stk::FileLoop silence("../stk/rawwaves/silence.raw", true);

    // Open a 16-bit, one-channel WAV formatted output file
    stk::FileWvOut output(filename, 1, stk::FileWrite::FILE_WAV, stk::Stk::STK_SINT16);

    const int note_tail = note_length / 8;
    const int note_padding = 0;
    const int file_padding = 10000;
    for (int note : song)
    {
        for (int i = 0; i < note_length; i++) {
            instrument.noteOn(scale[note], 0.8);
            output.tick(instrument.tick());
        }
        for (int i = 0; i < note_tail; i++) {
            // [0.0, 1.0) -- bigger means it stops faster
            instrument.noteOff(0.5);
            output.tick(instrument.tick());
        }
        for (int i = 0; i < note_padding; i++){}
            output.tick(silence.tick());
    }

    for (int i = 0; i < file_padding; i++)
        output.tick(silence.tick());

    output.closeFile();
}

#endif
```

9.2.6 examples_cpp/twinkle.cpp

```cpp
#include <algorithm>
#include <cmath>
#include <iostream>
#include <unordered_map>
#include <vector>

#include <stk/BlowHole.h>
#include <stk/Bowed.h>
#include <stk/Rhodey.h>
#include <stk/Wurley.h>

#include "music_utils.hpp"

int main()
{
    std::vector<int> intro { 1, 1, 5, 5, 6, 6, 5 };
    std::vector<int> chorus { 4, 4, 3, 3, 2, 2, 1 };

    std::vector<int> bridge(chorus.size());
    std::transform(chorus.begin(), chorus.end(), bridge.begin(), [](int x) -> int { return x + 1; });

    std::vector<int> song;
    std::copy(intro.begin(), intro.end(), std::back_inserter(song));
    std::copy(chorus.begin(), chorus.end(), std::back_inserter(song));
    std::copy(bridge.begin(), bridge.end(), std::back_inserter(song));
    std::copy(bridge.begin(), bridge.end(), std::back_inserter(song));
    std::copy(intro.begin(), intro.end(), std::back_inserter(song));
    std::copy(chorus.begin(), chorus.end(), std::back_inserter(song));

    render("twinkle-blowhole.wav", song, stk::BlowHole(8.0));
    render("twinkle-bowed.wav", song, stk::Bowed());
    render("twinkle-rhodey.wav", song, stk::Rhodey());
    render("twinkle-wurley.wav", song, stk::Wurley());

    return 0;
}
```

9.2.7 examples_nh/arabic.nh

```

// Rast (maqam) scale
key_sig = init key_signature{
  scale={ 261.63 293.66 320.38 349.23 392.00 440.00 480.02 }
}
tempo = 150

sth = e/2.0

fun Distort dur p = [ (dur/8.0) (7.0 * dur/8.0) ] :\
  [ (AddPitchNum (-1) p key_sig) p ]

opening = [ e e ] : [ 5 5 ] . Repeat 2 (Distort q (5@0)) .\
  [e q e q e e q ] : [ 4 5 4 3 4 3 2 ] . e:[2 3] . Distort e (4@0) .\
  Distort e (5@0) . Distort sth (4@0) . Distort sth (3@0) . e:2

bridge = [ e e sth sth e ] : [ 3 4 3 2 1 ] . Distort q (2@0) .\
  Distort e (3@0) . Distort e (2@0) . Distort q (1@0)

arabic = Relative 1 (opening . bridge)
Render (Parallel {arabic}) "arabic.wav"
```

9.2.8 examples_nh/pachelbel_canon.nh

```
// Pachelbel's Canon
key_sig = d_major

// bass line
bass = half:@(-1)[ 1@1  5 6 3 4 1 4 5 ]

// opening line
intro = half:(@1[ 3 2 1 ] . [ 7 6 5 6 7 ])
next_intro = EndWith (half:2) (Relative (-2) intro)


// verse 1
```

```
climb = quarter:[ 1 3 ]
fall = Reverse climb
slip = quarter:[ 5 4 ]
// it's all thirds!!!
verse_1 = climb . slip . fall . Relative (-2) (slip . fall) . (q:[ 1 5 ]) . Relative 3 climb . slip
next_verse_1 = fall . (q:[ 2 7 ]) . Octave 1 climb . (q:[ 5@1  5 ]) . \
        Relative 3 fall . Relative 2 fall . (q:[ 1  1@1 ]) . ([ (q+e) e ] : [ 1@1  7 ])

// verse 2
verse_2 = eighth:([ 1@1  7  1@1 1 7@(-1) 5 2 3 1  1@1  7 6 5 6 ] . @1[ 5 6 ])
// it's all scales!!!
climb = eighth : (Scale (1@0) (4@0))
fall = Reverse climb
next_verse_2 = EndWith (e:4@1) (Relative 3 fall) . Relative 6 fall . fall . \
        StartWith (e:2) (Relative 1 fall) . climb . e:[ 5 1 5 4 ] . \
        StartWith (e:3) (Relative 3 fall) . Relative 1 fall . e:[ 1  6@(-1)  6 7 ] . \
        Relative 4 fall . EndWith (e:6) fall . StartWith (e:5) (Relative 3 fall)

// bridge
qqh = Repeat 2 [ q q h ]
bridge = qqh:[ 3@(-1)  3 2 ~ 1 3 ] . h:@(-1)[ 6 5 6 7 ] . qqh:[ 1@1  1  7@(-1)  ~  6@(-1)  1 ] . \
        (h+q):1 . q:[ 1 1 4 2 5 ]
bridge = Octave 1 bridge

// chorus
rhythm = Repeat 2 [ e (e/2.0) (e/2.0) ] . Repeat 8 [ (e/2.0) ]
tones = @1[ 5 3 4 5 3 4 5 ] . Scale (5@0) (4@1) . \
@1[ 3 1 2 3 ] . [ 3 4 5 6 5 4 5 3 4 5 ] . \
[ 4 6 5 4 3 2 3 2 ] . Scale (1@0) (6@0) . \
[ 4 6 5 6 7 1@1 ] . Scale (5@0) (5@1) . \
@1[ 3 1 2 3 2 1 2  7@(-1)  1 2 3 2 1  7@(-1) ] . \
[ 1@1  6 7  1@1  1 2 3 4 3 2 3  1@1  7  1@1 ] . \
[ 6  1@1  7 6 5 4 5 4 ] . Scale (3@0) (1@1) . \
[ 6  1@1  7  1@1  7 6 7 ] . @1[ 1 2 1  7@(-1)  1 ] . [ 6 7 ]
chorus = Repeat 8 rhythm : tones

// entire song
```

```
melody = intro . next_intro . verse_1 . next_verse_1 . \
  verse_2 . next_verse_2 . bridge . chorus
first_part = melody . whole:1@1
second_part = (4.0*whole):~ . RemoveEnd (4.0*whole) melody . whole:3@1
bass_part = Extend (Length melody) bass . whole:5

canon =  Parallel { first_part second_part bass_part }
canon$volumes = { 1.0 1.0 0.5 }
Render canon "canon.wav"
```

9.2.9 examples_nh/row_your_boat.nh

```

// Row, Row, Row Your Boat
de = 1.5*e
se = e/2.0
rhythm = [ q q de se q de se de se h ] . Repeat 12 [ triplet ] . [ de se de se h ]
tune = [ 1 1 1 2 3 3 2 3 4 5 ] . Repeat 3 [ 1@1 ] . Repeat 3 [ 5 ] . Repeat 3 [ 3 ] . \
    Repeat 3 [ 1 ] . [ 5 4 3 2 1 ]

row_your_boat = rhythm : tune
Render (Parallel {row_your_boat}) "row_your_boat.wav"

// Row, Row, Row Alone (Sad Boat)
key_sig = c_minor
// the key signature is grabbed when the track object is created!
sad_boat = rhythm : tune
Render (Parallel {sad_boat}) "row_alone.wav"



// Roll, Roll, Roll Ya Rims (Gangsta Boat)
key_sig = f_minor
// make it sound dirty and dangerous
swag_boat = Octave (-1) (rhythm : tune)

// add a background track
sick_beat = Repeat 2 [ e e e (e/2.0) (e/2.0) ] . [ e e de se e e q ]
```

```
sick_tune = @1[ 1@1  5 6 6 5 4 4 5  1@1  1@1  1@1  5 6 5 4 4 ~ ]
sick_accomp = sick_beat : sick_tune . \
            EndWith (e:[ 7  1@1  ~ ]) (sick_beat : sick_tune)

gangsta_boat = Parallel { swag_boat sick_accomp }
gangsta_boat$volumes = { 1.0 0.35 }
Render gangsta_boat "roll_ya_rims.wav"
```

9.2.10 examples_nh/stairway_to_heaven.nh

```

tempo = 74

// stairway to heaven - led zeppelin
intro = eighth : [ 6@(-1)  1 3 6  7,5#  3 1 7 ] .\
  e : [ 1@1,5  3 1  1@1  4#,4#@(-1)  2  6@(-1)  4 ] .\
  e : [ 3,4@(-1)  1  6@(-1) ] . q:1 . e : [ 3 1  6@(-1) ]
fin_chord = 5@(-1),7@(-1)
fin = e:fin_chord,7@(-2) . Relative 1 ([ e (q+e) ]:fin_chord,5@(-2))
intro = intro . fin . Octave (-1) (e:[ 6@(-1) 4 3 ])

// note that the next phrase is the same except for the first and last notes
intro_next = EndWith ([ e e h ]:Chords fin . q:~) (StartWith (e:6@(-2)) intro)

stairway = intro . intro_next

all_the_way_to_heaven = Parallel { stairway }
Render all_the_way_to_heaven "stairway_to_heaven.wav"
```

9.2.11 examples_nh/twinkle.nh

```
// Twinkle
// main parts
intro = quarter:[ 1 1 5 5 6 6 ] . half:5
chorus = Rhythms intro : [ 4 4 3 3 2 2 1 ]
bridge = Relative 1 chorus

// the tune
```

```
twinkle_melody = intro . chorus . bridge . bridge . intro . chorus
twinkle_harmony = Relative 2 twinkle_melody

// supporting line
base = eighth:[ 1 5 3 5 ]
rise = eighth:[ 1 6 4 6 ]
fall = eighth:[ 7@(-1)  5 2 5 ]
bottom = eighth:[ 6@(-1)  5 1 5 ]

intro_accomp = base . base . rise . base
chorus_accomp = fall . base . bottom . base
bridge_accomp = base . fall . base . fall

// the accompaniment
accomp = intro_accomp . chorus_accomp . bridge_accomp . \
            bridge_accomp . intro_accomp . chorus_accomp
twinkle_bass = Octave (-1) accomp

// the song
twinkle = Parallel { twinkle_melody twinkle_harmony twinkle_bass }
twinkle$volumes = { 1.0 0.5 0.5 }
Render twinkle "twinkle.wav"
```

9.2.12 examples_nh/walk_this_way.nh

```

// walk this way - aerosmith
tempo = 144
time_sig = two_two

basic = [ 6 6# 7  3@1 ]
bass = Octave (-2) (e:basic . e:~ . [ e e e q ]:basic . [ q h ]:[ 3 ~])
bass = bass . EndWith ([ e e h ]:@(-2)[ 5 3 3@1 ]) bass .\
  bass . EndWith ((h+q):6@(-2)) bass

walk = [ h h (w-e) e ]:[ 7 7 7 ~ ] . [ h h e (w-e) ]:[ 6 6 7 5 ]
walk = Length bass : ~ . Repeat 2 walk
bass = Extend (Length walk) bass
```

```

walk_this_way = Parallel { walk bass }
walk_this_way$volumes = { 1.0 0.8 }
Render walk_this_way "walk_this_way.wav"
```

9.2.13 examples_nh/you_belong_with_me.nh

```

// You Belong With Me (Taylor Swift)
key_sig = fsharp_major

resolve = [ 1 2 3 ]
cont = [ 3 2 ]
open = [ 5 3 ] . cont
close = [ 5 ] . Repeat 2 cont
gf_upset = [ q e e q e e q q e q q ] : (open . [ 1 1 ] . close)
something_said = [ e e e e e q e q q q q ] : ([ 2 5 ] . open . close)
your_humor = [ e q e q e q q e q h h w ] : ([ 2 4 3 ] . resolve . cont . resolve . [ ~ ])
verse = Repeat 2 (gf_upset . something_said . your_humor)

rise = [ 6  1@1 ]
close = @1[ 2 1 ]
short_skirts = Repeat 2 (q:(rise . close))
cheer_captain = [ q q e e q q e e q q ] : (rise . [ 1@1 ] . close . OctaveChordList 1 resolve . close)
dreaming = [ e e e e q e e e e e q e e q ] : (rise . [ 1@1  6  2@1  5 5 ] . Repeat 3 rise . [ 6 ])
looking_for = [ e e q e e q e e e e e q e ] : ([ 6  1@1  6 ] . @1[ 3 2 2 1 1 3 2 1 1 2 ])
prechorus = short_skirts . cheer_captain . dreaming . looking_for

dq = q + e
understands = [ e e e q e q e q e e e q q ] : \
(@1[ 1 1 2 ] . Repeat 3 (OctaveChordList 1 cont) . [ 2 ] . ReverseList resolve)
been_here = [ q e q e q e q q q dq dq h ] : \
(Repeat 3 (OctaveChordList 1 cont) . ReverseList resolve . [ 4 3  6@(-1) ])
belong = [ e e e e q dq dq h e e q e h h ] : (@1[ 1 1 1 ] . [ 6 5 ] . @1[ 4 3 ] . [ 4 4 4 4 3 3  ~ ])
chorus = understands . been_here . belong

cap = e:@1[ 1 1 2 3 2 ]
driving = cap . [ e q ] : [ 1@1 6 ]
```

```
bridge = h:[ 4@1 ] . driving . driving . cap . [ e q ] : @1[ 1 1 ] . \
cap . [ e (q+e) (h+e) ] : @1[ 1 3 2 ]

build = [ e q ] : [ 6 1@1 ]
cant_you_see = e:6 . Repeat 3 build . e:@1[ 1 1 ] . [ e q q ]:resolve
belong_short = RemoveEnd (3.0*h+e) belong
chorus_out = cant_you_see . been_here . belong_short . been_here . belong

// all of tswift
taylor_swift = verse . prechorus . chorus . bridge . chorus_out
Render (Parallel { taylor_swift } ) "you_belong_with_me.wav"
```

9.2.14 install_dependencies.sh

```
#!/bin/bash

# Automates the OCaml installation described in:
# https://github.com/realworldocaml/book/wiki/Installation-Instructions
# ...plus some project-specific libraries.

# Safer shell scripting: https://sipb.mit.edu/doc/safe-shell/
set -euf -o pipefail

command_exists () {
    hash "$1" 2> /dev/null;
}

choice () {
    while true; do
        read -p "$1 [y/n] " yn
        case $yn in
            [Yy]* ) return 0;;
            [Nn]* ) return 1;;
            * ) echo 'Please answer yes or no.';;
        esac
    done
}

```

```
die () {
    echo "$1"
    exit 1
}

platform='unknown'
uname_str=`uname`
if [[ "$uname_str" == 'Linux' ]]; then
    platform='linux'
    linux_distro=`lsb_release -is`
    if [[ "$linux_distro" == 'Ubuntu' ]]; then
        platform='ubuntu'
    elif [[ "$linux_distro" == 'Debian' ]]; then
        platform='debian'
    else
        echo -ne 'Unsupported Linux distro: '
        lsb_release -is
        die 'Sorry :('
    fi
elif [[ "$uname_str" == 'Darwin' ]]; then
    platform='osx'
else
    die 'Unsupported platform. Sorry :('
fi

echo 'Downloading submodules'
git submodule update --init

echo 'Installing OPAM, OCaml, and STK'

opam_extra_args=""

if [[ "$platform" == 'ubuntu' || "$platform" == 'debian' ]]; then
    sudo apt-get update
    if ! apt-cache show opam > /dev/null; then
        echo 'Your Ubuntu version is too old. Adding ppa:avsm/ppa to get OPAM.'
        if ! command_exists 'add-apt-repository'; then
```

```
            sudo apt-get install python-software-properties
        fi
        sudo add-apt-repository ppa:avsm/ppa
        sudo apt-get update
    fi
    sudo apt-get install m4 ocaml-native-compilers camlp4-extra opam \
    stk stk-doc libstk0-dev

    # Ubuntu's OCaml package is ridiculously old. Ask OPAM to compile v4.02.3 for us.
    opam_extra_args="--comp 4.02.3"
elif [[ "$platform" == 'osx' ]]; then
    # Check for Xcode
    if ! command_exists 'xcode-select'; then
        die 'Please install Xcode from the Mac App Store.'
    fi

    # Check if command line tools installed
    if ! xcode-select -p > /dev/null; then
        xcode-select --install
        die 'Please install the Xcode command line tools and run this script again.'
    fi

    if command_exists 'brew'; then
        echo 'Installing packages with Homebrew.'
        brew install opam stk
    elif command_exists 'port'; then
        echo 'Installing packages with MacPorts.'
        sudo port install opam stk
    else
        die 'Please install Homebrew or MacPorts to get OPAM.'
    fi
fi

# Configure OPAM
if [[ -x "$HOME/.opam/" ]]; then
    echo 'opam init has already run. Skipping.'
else
```

```
    echo 'By default, OPAM modifies your .profile, .ocamlinit, and auto-complete scripts.'
    if choice 'Install OPAM with the default settings?'; then
        # Answer 'y' to all of OPAM's questions
        # For some reason, opam init doesn't return 0 on success.
        yes | opam init $opam_extra_args || true
    else
        # Let the user deal with OPAM
        opam init $opam_extra_args || true
    fi
fi

# Add OPAM stuff to current session
eval `opam config env`

echo 'Installing core and utop.'
opam install core utop

echo 'Please add the following to ~/.ocamlinit:'
echo ''
echo \
'#use "topfind";;
#thread;;
#camlp4o;;
#require "core.top";;
#require "core.syntax";;'
echo ''

echo 'Done! For editor-specific setup, please see this guide:'
echo 'https://github.com/realworldocaml/book/wiki/Installation-Instructions'
```

9.2.15 lib/array.nh

```
include std

extern "support.hpp" "nh_support" "shuffle" fun ShuffleInts int{} -> int{}
extern "support.hpp" "nh_support" "shuffle" fun ShuffleFloats float{} -> float{}

fun Size l = (
  //Returns the number of elements in list
  i = 0
  for item in l do (Ignore item; i = i + 1)
  i
)

fun ReverseList l = (
  //Returns an array with all the elements reversed
  size = Size l
  if size < 2 then
    l
  else (
    out = [(l.(size-1))];
    for i in Range 2 (size+1) do
      out = out . [l.(size - i)]
    out
  )
)

fun IsMember l x = (
  //Returns true if list contains element
  mem = false
  for item in l do mem = (mem || (item == x))
  mem
)


fun SameList x iter = (
  //Returns a list with x repeated iter amount of times
  out = {x}
  for i in Range 1 iter do (out = out . {x}; Ignore i)
  out
)

fun AppendCopy count list = (

)
```

```

fun Pop list = (

)

// returns an array of all ints in [low, high)
fun Range low high = if high <= low then int{} else { low } . Range (low + 1) high
```

9.2.16 lib/io.nh

```
include types

extern "support.hpp" "nh_support" "print_string" fun Print string -> unit

fun PrintEndline str = (
  //Prints the string followed by a newline to standard out
  Print str
  Print "\n"
)

fun PrintInt n = Print (StringOfInt n)

fun PrintFloat n = Print (StringOfFloat n)

fun PrintBool n = Print (StringOfBool n)
```

9.2.17 lib/math.nh

```
extern "support.hpp" "std" "pow" fun Pow float float -> float
extern "support.hpp" "std" "log" fun Log float -> float

extern "support.hpp" "std" "floor" fun Floor float -> float
extern "support.hpp" "std" "ceil" fun Ceiling float -> float

fun Min x y = (
  //Returns the minimum of the two elements
  if x <= y then
    x
  else
    y
)

fun Max x y = (
  //Returns the maximum of the two elements
  if x >= y then
    x
  else
    y
)
```

9.2.18 lib/std.nh

```
include array
include io
include math
include types

fun Ignore x = ()

const eighth = 0.125
const e = eighth
const quarter = 0.25
const q = quarter
const triplet = 0.25 / 3.0
const t = triplet
const half = 0.5
const h = 0.5
const whole = 1.0
const w = whole

type key_signature = {
  scale = { 261.63 293.66 329.63 349.23 392.00 440.00 493.88 }
}

const f3 = 174.61
const fs3 = 185.00
const g3 = 196.00
const gs3 = 207.65
const a3 = 220.00
const as3 = 233.08
const b3 = 246.94
const c4 = 261.63
const cs4 = 277.18
const d4 = 293.66
const ds4 = 311.13
const e4 = 329.63
const f4 = 349.23
const fs4 = 369.99
const g4 = 392.00
const gs4 = 415.30
const a4 = 440.00
const as4 = 466.16
const b4 = 493.88
const c5 = 523.25
const cs5 = 554.37
const d5 = 587.33
const ds5 = 622.25
const e5 = 659.25
const f5 = 698.46

const c_major = init key_signature
const f_minor = init key_signature{ scale={ f3 g3 gs3 as3 c4 cs4 ds4 } }
const c_minor = init key_signature{ scale={ c4 d4 ds4 f4 g4 gs4 as4 } }
const fsharp_major = init key_signature{ scale={ fs3 gs3 as3 b3 cs4 ds4 f4 } }
const d_major = init key_signature{ scale={ d4 e4 fs4 g4 a4 b4 cs5 } }
const g_major = init key_signature{ scale={ g3 a3 b3 c4 d4 e4 fs4 } }

type time_signature = {
  upper = 4
  lower = 4
}

const two_two = init time_signature{ upper=2; lower=2 }
```

```

type pitch = {
  rank = 1
  octave = 0
  offset = 0
}

type chord = {
  pitches = pitch{}
}

//Returns a chord containing the given pitch
fun ChordOfPitch p = init chord { pitches = {p} }

fun PitchOfInt i = (
  //Returns a pitch with the given rank
  if (i < 0) || (i > (Size key_sig$scale)) then (
    throw "Could not construct pitch from int";
    init pitch)
  else
    init pitch { rank = i }
)

fun NormalizePitch p keysig = (
  //Returns a valid pitch with a rank between 0 and the length of the key signature
  scale_size = Size keysig$scale
  if p$rank < 1 then (
    octave_offset = (-1 * p$rank) / scale_size + 1
    p$rank = p$rank + octave_offset * scale_size
    p$octave = p$octave - octave_offset
  ) else (
    p$rank = p$rank - 1 // Make normalization easier. Pitch is [1, 7] -> [0, 6]
    p$octave = p$octave + p$rank / scale_size
    p$rank = p$rank % scale_size
    p$rank = p$rank + 1 // Undo our change: [0, 6] -> [1, 7]
  )
  p
```

```
)

//Returns a pitch shifted up by the given octave
fun AddPitchOctave p octaves = ( p$octave = p$octave + octaves; p)

fun AddPitchNum num p keysig = (
  //Returns a pitch object shifted up by num for the given p and key signature
  p$rank = p$rank + num
  NormalizePitch p keysig
)

fun AddChordNum num ch keysig = (
  //Returns a chord with all the notes of ch shifted up by num for the given key signature.
  out_pitch = pitch{}
  for p in ch$pitches do
    out_pitch = out_pitch . { (AddPitchNum num p keysig) }
  ch$pitches = out_pitch
  ch
)

fun PrintChord ch = (
  //Prints each pitch of chord
  size = Size ch
  for c in ch do (
    x = c$pitches
    for p in x do
      Print (StringOfInt p$rank)
    Print "\n"
  )
)

fun PrintRhythms rh = (
  //Prints each duration inside rh
  size = Size rh
  for r in rh do (
    Print (StringOfFloat r)
  )
```

```
)


type track = {
  key_sig = init key_signature
  time_sig = init time_signature
  tempo = 120
  chords = chord{}
  durations = float{}
  // currently we can only assign a single volume to the track
  // of dynamics control we lack
  // yes, it’s a simplicity hack
  // but i’d like to hit the sack
  volume = 1.0
}

type song = {
  tracks = track{}{}
  volumes = float{}
}

//Flats the pitch
fun FlatPitch p = ( p$offset = p$offset - 1; p )
//Sharps the pitch
fun SharpPitch p = ( p$offset = p$offset + 1; p )

fun ChordOfChords c1 c2 = (
  //Returns a chord object containing the union of pitches in both chords
  c = init chord
  for p in c1$pitches . c2$pitches
    do if IsMember c$pitches p then () else c$pitches = c$pitches . {p}
  c
)

fun Rest = init chord

fun ConcatTracks t1 t2 = (
  // Returns a new track with the tracks arranged sequentially
  //(use the . operator instead)
  if t1$key_sig == t2$key_sig &&\
      t1$time_sig == t2$time_sig &&\
      t1$tempo == t2$tempo
  then (
    t1$chords = t1$chords . t2$chords
    t1$durations = t1$durations . t2$durations
    t1
  )
  else (throw "Cannot concat tracks with different key or time signature or tempo"; t1)
)

key_sig = init key_signature
time_sig = init time_signature
tempo = 120

fun ZipSame f c = (
  if (Size f) != (Size c) then
    (throw "Cannot Zip Arrays of Different Lengths"; init track)
  else(
    init track{chords = c; durations = f;key_sig = key_sig
               time_sig = time_sig; tempo = tempo}
  )
)

fun ZipDiff f c = (
  out_f = f
  out_c = c
  if (Size f) == 1 then(
    iter = Size c
    out_f = SameList f.(0) iter
  )
  else (
    iter = Size f
    out_c = SameList c.(0) iter
  )
  init track{chords = out_c; durations = out_f; key_sig = key_sig
              time_sig = time_sig; tempo = tempo}
)

fun Scale pitch_start pitch_end = (
  //Returns an array of chords representing the scale in the current key signature
  //between pitch_start and pitch_end
  c = chord{}
  oct_diff = pitch_end$octave - pitch_start$octave
  pitch_diff = pitch_end$rank - pitch_start$rank
  total_diff = (Size key_sig$scale) * oct_diff + pitch_diff
  direction = be 1 unless total_diff < 0 inwhichcase -1
  for x in Range 0 (direction * total_diff + 1) do
    c = c . [(AddPitchNum (direction*x) pitch_start key_sig)]
  c
)

extern "support.hpp" "nh_support" "render_impl" fun RenderImpl float{}{}{} float{}{} float{} string -> unit

fun Render song filename = (
  //Creates a WAV file of the song
  if Size song$tracks != Size song$volumes then throw "Internal error: song has mismatched tracks and volumes" else ()
  freqs = float{}{}{}
  durs = float{}{}
  for track_chan in song$tracks do (
    freq_chan = float{}{}
    dur_chan = float{}
    for track in track_chan do (
      freq_chan = freq_chan . FrequenciesOfChords track$chords track$key_sig
      dur_chan = dur_chan . SecondsOfDurations track$durations track$time_sig track$tempo
    )
    freqs = freqs . { freq_chan }
    durs = durs . { dur_chan }
  )
  RenderImpl freqs durs song$volumes filename
)
```

```
fun ChromaticOfFrequency base_frequency frequency = (
  //Computes the offset of frequency relative to base_frequency
  Log (frequency/base_frequency) / Log 2.0 * 12.0
)

fun FrequencyOfChromatic base_frequency chromatic = (
  //Computes the frequency for the given chromatic
  base_frequency * Pow 2.0 (chromatic/12.0)
)

fun OffsetFrequency frequency offset = (
  //Returns the frequency for the given offset
  // this function should be overridden by the user if using non chromatic scales
  // how a flat or sharp affects the music helps tell a different tale
  base_freq = 440.0
  FrequencyOfChromatic base_freq (ChromaticOfFrequency base_freq frequency + offset)
)

fun FrequencyOfPitch pitch keysig = (
  //Returns the frequency of the pitch for the given key signature
  index = pitch$rank - 1
  frequency = keysig$scale.(index) * Pow 2.0 (FloatOfInt pitch$octave)
  OffsetFrequency frequency (FloatOfInt pitch$offset)
)

fun FrequenciesOfChord chord keysig = (
  //Returns a list representing the frequencies of the chord
  if Size chord$pitches == 0 then { 0.0 }
  else (
    freqs = float{}
    for pitch in chord$pitches do (
      freqs = freqs . { (FrequencyOfPitch pitch keysig) }
    )
    freqs
  )
)
```

```
fun FrequenciesOfChords chords keysig = (
  //Returns a list representing the frequencies of the chords
  freqs = float{}{}
  for chord in chords do (
    freqs = freqs . { (FrequenciesOfChord chord keysig) }
  )
  freqs
)

fun SecondsOfDurations durations timesig tempo = (
  //Returns a list consisting of the durations represented as seconds
  multiplier = FloatOfInt timesig$lower * 60.0 / FloatOfInt tempo
  seconds = float{}
  for duration in durations do (
    seconds = seconds . { (duration * multiplier) }
  )
  seconds
)

fun Arpeggio chord = (
  //Returns an array of chords representing the arpeggio using the pitches from chord
  chord$pitches
)

fun Rhythms track = (
  //Returns the array of note durations of the track
  track$durations
)

fun Chords track = (
  // Returns the array of chords of the track
  track$chords
)

fun Parallel ts = (
  //Returns a song object with the tracks aligned in parallel (to be played concurrently)
  s = init song
  for t in ts do (
    s$tracks = s$tracks . { {t} }
    s$volumes = s$volumes . { 1.0 }
  )
  s
)

fun Sequential ts = (
  //Returns a song object with the tracks aligned in a single sequence (to be played sequentially)
  init song { tracks = {ts} }
)

fun Length track = (
  //Returns the duration of the track
  length = 0.0
  for d in track$durations do length = length + d
  length
)

fun Extend length tr = (
  //Returns a track object that repeats the given track for length units. If len is not an
  //even multiple of the Length tr, the remainder is padded with rests
  multiplier = length / Length tr
  // Add as many copies of the track as will fit
  orig_chords = tr$chords; orig_durations = tr$durations
  for i in Range 0 (IntOfFloat multiplier - 1) do (
    Ignore i
    tr$chords = tr$chords . orig_chords
    tr$durations = tr$durations . orig_durations
  )
  // Fill the rest of the space with a rest
  pad = length - Length tr
  if pad > 0.0 then (
    tr$chords = tr$chords . { (init chord) }
    tr$durations = tr$durations . [ pad ]
  ) else ()
  tr
```

```
)

fun StartWith tr base_track = (
  //Returns a track with the start of base_track replaced with tr
    Reverse (EndWith (Reverse tr) (Reverse base_track))
)

fun EndWith tr base_track = (
  //Returns a track with the end of base_track replaced with tr
  RemoveEnd (Length tr) base_track . tr
)

fun Reverse tr = (
  //Returns the track with all the notes reversed
  init track{time_sig = tr$time_sig; key_sig = tr$key_sig; tempo=tr$tempo
  chords = (ReverseList tr$chords); durations = (ReverseList tr$durations)}
)

fun OctaveChordList shift ch = (
  //Returns the chord object shifted by num octaves
  rel_chords = chord{}
  for c in ch do
    rel_chords = rel_chords . {(AddChordNum (shift * Size key_sig$scale) c key_sig)}
  rel_chords
)

fun Relative shift tr = (
  //Returns a track with all the notes shifted up by shift
  rel_chords = chord{}
  for c in tr$chords do
    rel_chords = rel_chords . {(AddChordNum shift c tr$key_sig)}
  tr$chords = rel_chords
  tr
)

fun Octave shift tr = (
  //Returns the track object shifted by shift octaves.
  Relative (shift*(Size tr$key_sig$scale)) tr
)

fun Repeat times tr = (
  //Returns a track object with the given track repeated times number of times
  if times <= 1 then
    tr
  else
    tr . Repeat (times - 1) tr
)

fun RemoveEnd len tr = (
  //Returns a track with the last $len$ duration of $tr$ sliced off
  dur = (Length tr) - len
  if dur < 0.0 then
    (throw "Remove End length is too long"; tr)
  else(
    notes = tr$chords
    durations = tr$durations
    out_chords = chord{}
    out_dur = float{}
    for i in Range 0 (Size notes) do (
      if dur <= 0.0 then
        ()
      else(
        to_add = durations.(i)
        out_dur = out_dur . {(Min dur to_add)}
        dur = dur - to_add
        out_chords = out_chords . {notes.(i)}
      )
    )
    tr$durations = out_dur
    tr$chords = out_chords
    tr
  )
)

```

```
// TODO: volume mix functions for song objects
```

9.2.19 lib/types.nh

```
extern "support.hpp" "nh_support" "int_of_float" fun IntOfFloat float -> int
extern "support.hpp" "std" "to_string" fun StringOfFloat float -> string

extern "support.hpp" "nh_support" "float_of_int" fun FloatOfInt int -> float
extern "support.hpp" "std" "to_string" fun StringOfInt int -> string

fun StringOfBool n = if n then "true" else "false"
```

9.2.20 README.md

```
# ♫#ïÿŔâČč
A programming language for exploring and creating music

# Authors
- [Kevin Chen](http://kevinchen.co/)
- Brian Kim
- Edward Li

# Compiling

Run the ‘install_dependencies.sh‘ script to set up submodules, install OCaml, and install third-party libraries.

This script is maintained on Ubuntu 15.04 and OS X 10.11 El Capitan, although it will probably work on Ubuntu 14.x,
OS X 10.10 Yosemite, and Debian as well.

Once you have the dependencies, just ‘cd‘ into the directory you want and ‘make‘.

# Contributing

1. Grab the lastest master: ‘git checkout master && git pull origin master‘
2. Create your branch: ‘git checkout -b alice_feature_name‘
3. Make some changes: ‘git commit ...‘
4. Squash commits: ‘git rebase -i master‘, then change everything except the first commit to ‘squash‘
5. Rebase on the latest master: ‘git checkout master && git pull origin master && git rebase master alice_feature_name‘
```

```
6. Run tests: ‘make test‘
7. Push for code review: ‘git push -f origin alice_feature_name‘ (only use push -f on feature branches, not master)

# Syntax Highlighting

Syntax highlighting is provided for Sublime Text. In Sublime, go to:
Preferences -> Browse Packages -> User
and copy the file (in style/)
note-hashtag.tmLanguage
into that directory.
```

9.2.21 src/ast.ml

```ocaml
open Core.Std

type type_name = string
type t =
  | Unit
  | Int
  | Float
  | String
  | Bool
  | Type of type_name
  | Array of t

type binary_operator =
  | Add | Sub | Mul | Div | Mod
  | Eq | Neq | Lt | Lte
  | And | Or | Zip
  | Concat | Chord | Octave

type unary_operator =
  | Not
  | Neg
  | Sharp
  | Flat

type var_reference = string list

type mutability = Mutable | Immutable

type expr =
  | Binop of expr * binary_operator * expr
  | Uniop of unary_operator * expr
  | LitBool of bool
  | LitInt of int
  | LitFloat of float
  | LitStr of string
  | VarRef of var_reference
  | FunApply of string * expr list
  | ArrIdx of var_reference * expr
  | Arr of expr list * t option
  | ArrMusic of expr list
  | Block of expr list
  | Conditional of expr * expr * expr
  | For of string * expr * expr
  | Throw of expr
  | Assign of var_reference * expr * mutability
  | StructInit of string * expr list

type fundef =
  | FunDef of string * string list * expr

type externfun =
  (* Header file name, namespace, C++ function name, NH function name, list of param types, return type *)
  | ExternFunDecl of string * string * string * string * t list * t

type typedef =
  | TypeDef of string * expr list

type program = string list * fundef list * externfun list * expr list * typedef list

let rec string_of_type t =
  match t with
  | Unit -> "unit"
  | Int -> "int"
  | Float -> "float"
  | String -> "string"
  | Bool -> "bool"
  | Type(name) -> name
  | Array(t) -> string_of_type t ^ "{}"

let string_of_type_op t =
  match t with
  |Some t -> string_of_type t
  |None -> "None"

let string_of_op o =
  match o with
  | Add -> "+"
  | Sub -> "-"
  | Mul -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Eq  -> "=="
  | Neq -> "!="
  | Lt  -> "<"
  | Lte -> "<="
  | And -> "&&"
  | Or  -> "||"
  | Concat -> "."
  | Chord -> ","
  | Zip -> ":"
  | Octave -> "@"

let string_of_unop o = match o with | Not -> "!" | Neg -> "-" | Sharp -> "#"
                                    | Flat -> "b"

let rec string_of_expr e =
  match e with
  | Block l -> String.concat ~sep:" " [ "["; string_of_exp_list l; "]" ]
  | Conditional(x, y, z) ->
```

```
      String.concat ~sep:" " [ "IF"; string_of_expr x; "THEN"; string_of_expr y; "ELSE"; string_of_expr z ]
  | For(x, y, z) -> String.concat ~sep:" " [ "FOR"; x; "IN "; string_of_expr y; "DO"; string_of_expr z ]
  | Binop(x, op, y) -> String.concat ~sep:" " [ "("; string_of_expr x; string_of_op op; string_of_expr y; ")" ]
  | Uniop(op, x) -> String.concat ~sep:" " [ "("; string_of_unop op; string_of_expr x; ")" ]
  | LitBool(x) -> Bool.to_string x
  | LitInt(x) -> Int.to_string x
  | LitFloat(x) -> Float.to_string x
  | LitStr(x) -> x
  | VarRef(x) -> String.concat ~sep:"$" x
  | Assign(name, expr, mutability) ->
      sprintf "( %s %s = %s )"
        (if mutability = Immutable then "let" else "var") (string_of_expr (VarRef(name))) (string_of_expr expr)
  | StructInit(x, y) -> String.concat ~sep:" " [ x; string_of_exp_list y ]
  | FunApply(x, y) -> String.concat ~sep:" " [ x; "("; string_of_exp_list y; ")" ]
  | ArrIdx (x, y) -> String.concat ~sep:" " [ string_of_expr (VarRef(x)); ".("; string_of_expr y; ")" ]
  | Arr(x, t) -> String.concat ~sep:" " [ "{"; string_of_exp_list x; "}, "; string_of_type_op t ]
  | ArrMusic(x) -> String.concat ~sep:" " [ "["; string_of_exp_list x; "]"]
  | Throw(x) -> String.concat ~sep:" " ["Throw"; string_of_expr x]
and string_of_exp_list l =
  match l with
  | [] -> ""
  | [ s ] -> string_of_expr s
  | s :: rest -> String.concat ~sep:" " [ string_of_expr s; ","; string_of_exp_list rest ]

let string_of_fdef fdef =
  let FunDef(name, args, body) = fdef in
  String.concat ~sep:" " ([ name ] @ args @ [ string_of_expr body ])

let string_of_extern extern =
  let ExternFunDecl(hpp, ns, cpp_name, nh_name, param_types, ret_type) = extern in
    let cpp_path = String.concat ~sep:"::" [ hpp; ns; cpp_name ] in
    let type_strs = List.map param_types ~f:string_of_type in
    String.concat ~sep:" " ([ "extern"; cpp_path] @ type_strs @ ["->"; string_of_type ret_type; "as"; nh_name ])

let rec string_of_incl_list p =
  match p with
  | [] -> "[]"
  | s :: rest -> String.concat ~sep:" " [ s; string_of_incl_list rest ]

let rec string_of_typedefs typedefs =
  match typedefs with
  | [] -> ""
  | TypeDef(name, exprs) :: rest -> name ^ string_of_exp_list exprs ^ "\n" ^ string_of_typedefs rest

let string_of_prog_struc p =
  match p with
  | (incls, fdefs, externs, exprs, typedefs) ->
      String.concat ~sep:"\n"
        [ "INCLUDES: " ^ string_of_incl_list incls;
          "TYPEDEFS: " ^ string_of_typedefs typedefs;
          "FDEF: " ^ String.concat ~sep:"\n" (List.map fdefs ~f:string_of_fdef);
          "EXTFUN: " ^ String.concat ~sep:"\n" (List.map externs ~f:string_of_extern);
          "EXPR: " ^ string_of_exp_list exprs;
        ]
```

9.2.22 src/cast.ml

```
open Core.Std

open Ast

type binary_operator = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | And | Or
type unary_operator = Not | Neg

type decl = Ast.t * string


type expr =
  | LitUnit
  | LitBool of bool
  | LitInt of int
  | LitFloat of float
  | LitStr of string
  | InitList of expr list
  | Decl of decl
  | DeclAssign of decl * expr
  | VarRef of var_reference
  | Idx of var_reference * expr
  | Binop of expr * binary_operator * expr
  | Uniop of unary_operator * expr
  | Assign of var_reference * expr
  | Call of callable * expr list
  | Noexpr

and stmt =
  | Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | ForRange of decl * expr * stmt
  | While of expr * stmt

and callable =
  | Struct of string
  | Function of string * string * Ast.t list
  | Method of var_reference * string
  | LambdaRefCap of decl list * Ast.t * stmt list

type func_decl = {
  fnamespace : string;
  fname : string;
  fargs : decl list;
  treturn : Ast.t;
  body : stmt list;
}

type struct_decl = {
  sname: string;
  sargs: decl list;
}

```

```
type incl =
  | IncludeAngleBrack of string
  | IncludeQuote of string

type signature =
  | SigFunc of string * Ast.t * decl list
  | SigStruct of string

type program = incl list * signature list * decl list * struct_decl list * func_decl list

let sep = ";\n"
let ns = "::"

let wrap_static_cast expr to_type =
  Call(Function("", "static_cast", [ to_type ]), [ expr ])

let rec string_of_type t =
  match t with
  | Unit -> "unit_t"
  | Int -> "int64_t"
  | Float -> "double"
  | String -> "std" ^ ns ^ "string"
  | Bool -> "bool"
  | Type(type_name) -> type_name
  | Array(t) -> "std" ^ ns ^ "vector" ^ string_of_type_args [ t ]
and string_of_type_args args =
  if args = [] then ""
  else
    let string_of_type t = (match t with Ast.Type(_) -> "struct " | _ -> "") ^ string_of_type t in
    "<" ^ String.concat ~sep:", " (List.map args ~f:string_of_type) ^ ">"

let rec string_of_expr = function
  | LitUnit -> "LIT_UNIT"
  | LitBool(x) -> Bool.to_string x
  | LitInt(x) -> Int.to_string x
  | LitFloat(x) -> sprintf "%.17F" x
  | LitStr(x) -> "\"" ^ String.escaped x ^ "\""
```

```
 | InitList(exprs) -> "{ " ^ String.concat ~sep:", " (List.map exprs ~f:string_of_expr) ^ " }"
 | Decl(t, name) -> string_of_type t ^ " " ^ name
 | DeclAssign((t, name), e) -> string_of_type t ^ " " ^ name ^ " = " ^ string_of_expr e
 | VarRef(names) -> String.concat ~sep:"." names
 | Idx(name, e) -> string_of_expr (VarRef(name)) ^ "[" ^ string_of_expr e ^ "]"
 | Binop(e1, o, e2) ->
   let op_str =
    match o with
    | Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
    | Equal -> "==" | Neq -> "!="
    | Less -> "<" | Leq -> "<=" | Mod -> "%" | And -> "&&" | Or -> "||"
   in
   sprintf "(%s) %s (%s)" (string_of_expr e1) op_str (string_of_expr e2)
 | Uniop(o, e) -> (match o with Not -> "!" | Neg -> "-") ^ string_of_expr e
 | Assign(v, e) -> string_of_expr (VarRef(v)) ^ " = " ^ string_of_expr e
 | Call(callexpr, args) -> string_of_callable callexpr ^ "(" ^
   String.concat ~sep:", " (List.map args ~f:string_of_expr) ^ ")"
 | Noexpr -> ""

and string_of_callable = function
 | Struct(name) -> name
 | LambdaRefCap(decls, treturn, stmts) -> "[&] (" ^
   String.concat ~sep:", " (List.map decls ~f:(fun (t, name) -> string_of_type t ^ " " ^ name)) ^
   ") -> " ^ string_of_type treturn ^" "^ string_of_stmt (Block stmts)
 | Method(oname, fname) -> string_of_expr (VarRef(oname)) ^ "." ^ fname
 | Function(namespace, fname, tmpl_args) ->
   namespace ^ (if namespace <> "" then ns else "") ^ fname ^ string_of_type_args tmpl_args

and string_of_stmt = function
 | Block(stmts) -> "{\n" ^ String.concat (List.map stmts ~f:string_of_stmt) ^ "}\n"
 | Expr(expr) -> string_of_expr expr ^ sep
 | Return(expr) -> "return " ^ string_of_expr expr ^ sep
 | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
 | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
 | For(e1, e2, e3, s) ->
   "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^ string_of_expr e3  ^ ") " ^ string_of_stmt s
 | ForRange(rdecl, rexpr, s) ->
```

```
      "for (" ^ string_of_expr (Decl(rdecl)) ^ " : " ^ string_of_expr rexpr ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_fdecl fdecl =
  (* Return type *)
  (if fdecl.fname <> "main" then string_of_type fdecl.treturn else "int") ^ " " ^
  (* Namespace :: name *)
  (if String.is_empty fdecl.fnamespace then fdecl.fname else fdecl.fnamespace ^ ns ^ fdecl.fname) ^
  (* Arguments *)
  "(" ^ String.concat ~sep:", " (List.map fdecl.fargs ~f:(fun (t, name) -> string_of_type t ^ " " ^ name)) ^ ")\n{\n" ^
  (* Body statements *)
    String.concat (List.map fdecl.body ~f:string_of_stmt) ^
  "}\n"

let string_of_sdecl sdecl =
  "struct " ^ sdecl.sname ^ " {\n" ^
    String.concat ~sep:"\n" (List.map sdecl.sargs ~f:(fun decl -> string_of_stmt (Expr(Decl(decl))))) ^
    (* constructor *)
    "\n" ^ sdecl.sname ^ "(" ^
    String.concat ~sep:", " (List.map sdecl.sargs ~f:(fun decl -> string_of_expr (Decl(decl)))) ^
    ") : " ^ String.concat ~sep:", " (List.map sdecl.sargs ~f:(fun (_,n) -> n^"("^n^")")) ^
    " {}" ^ "\n"^
    (* empty constructor for temporary global initialization *)
    sdecl.sname ^ "(){}"
    ^"};\n"

let string_of_signature = function
  | SigStruct(name) -> "struct " ^ name
  | SigFunc(name, tret, decls) ->
      string_of_type tret ^ " " ^ name ^ "(" ^
      String.concat ~sep:", " (List.map decls ~f:(fun (t,_) -> string_of_type t)) ^
      ")"

let string_of_incl incl =
  match incl with
  | IncludeAngleBrack(path) -> "#include <" ^ path ^ ">"
  | IncludeQuote(path) -> "#include \"" ^ path ^ "\""
```

```

let string_of_program (incls, signatures, decls, structs, funcs) =
  String.concat (List.map incls ~f:(fun incl -> string_of_incl incl ^ "\n")) ^
  String.concat (List.map signatures ~f:(fun signature -> string_of_signature signature ^ ";\n")) ^
  String.concat (List.map structs ~f:(fun sdecl -> string_of_sdecl sdecl ^ "\n")) ^
  String.concat (List.map decls ~f:(fun decl -> string_of_stmt (Expr(Decl(decl))) ^ "\n" )) ^
  String.concat (List.map funcs ~f:(fun fdecl -> string_of_fdecl fdecl ^ "\n"))
```

9.2.23 src/cpp_sast.ml

```
open Core.Std
open Ast
open Sast
open Cast

let rec castx_of_sastx texpr =
  let unit_ret = Cast.Return(Cast.LitUnit) in
  let (expr, t) = texpr in
  match expr with
    | Sast.LitBool(x) -> Cast.LitBool(x)
    | Sast.LitInt(x) -> Cast.LitInt(x)
    | Sast.LitFloat(x) -> Cast.LitFloat(x)
    (* Comparison of string literals in C++ is undefined (you are actually comparing the two char* values) *)
    | Sast.LitStr(x) -> Cast.Call(Function("std", "string", []), [ Cast.LitStr(x) ])
    | Sast.LitUnit -> Cast.LitUnit

    | Sast.Binop(lexpr, op, rexpr) ->
        begin match op with
          | Ast.Concat ->
              Cast.Call(Cast.Function("nh_support","concat", []),[castx_of_sastx lexpr; castx_of_sastx rexpr])
          | Ast.Zip | Ast.Chord | Ast.Octave ->
              failwith (sprintf "Internal error: binop '%s' should have been converted to Call NhFunction in ast2sast"
                (Ast.string_of_op op))
          | _ as cop -> let op = begin match cop with
            | Ast.Add -> Cast.Add
            | Ast.Sub -> Cast.Sub
            | Ast.Mul -> Cast.Mult
            | Ast.Div -> Cast.Div
        | Ast.Mod -> Cast.Mod
        | Ast.Eq -> Cast.Equal
        | Ast.Neq -> Cast.Neq
        | Ast.Lt -> Cast.Less
        | Ast.Lte -> Cast.Leq
        | Ast.And -> Cast.And
        | Ast.Or -> Cast.Or
        | _ -> failwith "Internal error: failed to match all possible binops in sast2cast"
       end in Cast.Binop(castx_of_sastx lexpr, op, castx_of_sastx rexpr)
      end


    | Sast.Uniop(op, expr) ->
      begin match op with
       | Ast.Not -> Cast.Uniop(Cast.Not, castx_of_sastx expr)
       | Ast.Neg -> Cast.Uniop(Cast.Neg, castx_of_sastx expr)
       | _ -> failwith
         "Internal error: uniop flat and sharp should have been converted to Call NhFunction in ast2sast"
      end

    | Sast.VarRef(names) -> Cast.VarRef(names)

    | Sast.FunApply(fname, exprs) ->
      let (ns, fn) = match fname with
       | NhFunction(name) -> "", name
       | CppFunction(_, ns, name) -> ns, name
      in
      (* Convert args to C++ AST and wrap them in a static_cast. This resolves ambiguity when calling certain
       * overloaded functions:
       *   void f(int64_t a, int64_t b) {} // 1
       *   void f(double a, double b) {}   // 2
       *   f(1, 2)
       * Because the int arguments could be promoted to int64_t OR double, both functions are candidates in the C++
       * compiler's eyes. *)
      let exprs = List.map exprs ~f:(fun (e, t) -> let e = castx_of_sastx (e, t) in Cast.wrap_static_cast e t) in
      Cast.Call(Cast.Function(ns, fn, []), exprs)
    | Sast.ArrIdx(varname, expr)
      -> Cast.Idx(varname, castx_of_sastx expr)

    | Sast.Arr(exprs, ast_t) ->
      Cast.Call(Function("std", "vector", [ ast_t ]), [Cast.InitList(List.map exprs ~f:(castx_of_sastx) )])

    | Sast.Block(exprs) ->
      begin match List.rev exprs with
        | [] -> failwith "Internal Error: Sast.Block found to be empty when converting to cast"
        | ret_expr :: body ->
            (* cannot return assignments; pad with lit unit *)
            let (ret_expr,body) =
              match ret_expr with
                | Sast.Init(_, _, _),_ -> (Sast.LitUnit, Ast.Unit), ret_expr::body
                | _ -> ret_expr, body
            in
            Cast.Call(Cast.LambdaRefCap([], t, List.rev begin
              Cast.Return(castx_of_sastx ret_expr) ::
              List.map body ~f:(fun expr -> Cast.Expr(castx_of_sastx expr)) end), [])
      end

    | Sast.Conditional(condition, case_true, case_false) ->
      let condition = castx_of_sastx condition in
      (* Save the type before we throw it away *)
      let (_, ret_t) = case_true in
      (* Throw the expression into a block if it's not already a block *)
      let wrap_nonblock sexpr =
        match sexpr with
        | Sast.Block(_), _ -> sexpr
        | _, t -> Sast.Block([ sexpr ]), t
      in
      let case_true = castx_of_sastx (wrap_nonblock case_true)
      and case_false = castx_of_sastx (wrap_nonblock case_false) in
      (* Create if statement *)
      let if_stmt = Cast.If(condition, Cast.Return(case_true), Cast.Return(case_false)) in
      (* No need for Cast.Block because castx_of_sastx wraps each case in a block in a lambda *)
      Cast.Call(Cast.LambdaRefCap([], ret_t, [ if_stmt ]), [])
```

```

  | Sast.For((var_name, var_t), rexpr, dexpr) ->
    let for_stmt = Cast.ForRange((var_t, var_name), castx_of_sastx rexpr, Cast.Expr(castx_of_sastx dexpr)) in
    Cast.Call(Cast.LambdaRefCap([], Ast.Unit, [ for_stmt; unit_ret ]), [])

  | Sast.Exit(code) ->
    let cast_exit = Cast.Call(Cast.Function("", "exit", []),[Cast.LitInt(code)]) in
    Cast.Call(Cast.LambdaRefCap([], Ast.Unit, [Cast.Expr(cast_exit); unit_ret]), [])

  | Sast.Init(name, expr, _) ->
    let (_,t) = expr in Cast.DeclAssign((t, name), castx_of_sastx expr)

  | Sast.Assign(varname, expr) -> Cast.Call(
    (* assign and then return unit *)
    Cast.LambdaRefCap([], Ast.Unit,
     [Cast.Expr(Cast.Assign(varname, castx_of_sastx expr));
     Cast.Return(Cast.LitUnit)]),[])

  | Sast.Struct(typename, fields) ->
   let fields = List.sort fields ~cmp:(fun (ln,_) (rn,_) -> compare ln rn) in
   let args = List.map fields ~f:(fun (_,expr) -> castx_of_sastx expr) in
   Cast.Call(Cast.Struct(typename), args)

let castfun_of_sastfun fundef =
 let Sast.FunDef(fname, fargs, texpr) = fundef in
 let (_, return_type) = texpr in
 {
  fnamespace = "";
  fname = fname;
  fargs = List.map fargs ~f:(fun (s,t) -> (t,s));
  treturn = return_type;
  body = [ Cast.Return (castx_of_sastx (Sast.Block([texpr]),return_type)) ];
 }

let nequality_fun (Sast.TDefault(name, _)) =
 let body = [Cast.Block([Cast.Return(Cast.Uniop(Cast.Not,
  Cast.Call(Cast.Function("","operator==", []), [Cast.VarRef(["lhs"]); Cast.VarRef(["rhs"])])))])] in
  {
    fnamespace = "";
    fname = "operator!=";
    fargs = [(Ast.Type(name), "lhs"); (Ast.Type(name), "rhs")];
    treturn = Ast.Bool;
    body = body;
  }

let equality_fun (Sast.TDefault(name, fields)) =
  let struct_equal_fun fields =
    (* makes a condition for each field, "and"-ing the result *)
    let b = Cast.LitBool(true) in
    let conds = List.fold_left fields ~init:b ~f:(fun b (name, _) ->
      Cast.Binop(b, Cast.And, Cast.Binop(Cast.VarRef(["lhs";name]), Cast.Equal, Cast.VarRef(["rhs";name]))))
    in
    Cast.Block([Cast.Return(conds)])
  in
  let body = [ struct_equal_fun(fields)] in
  {
    fnamespace = "";
    fname = "operator==";
    fargs = [(Ast.Type(name), "lhs"); (Ast.Type(name), "rhs")];
    treturn = Ast.Bool;
    body = body;
  }

let casttype_of_sasttype (Sast.TDefault(name, fields)) =
  let fields = List.sort fields ~cmp:(fun (ln,_) (rn,_) -> compare ln rn) in
  let sargs = List.map fields ~f:(fun (n, (_,t)) -> (t,n)) in
  {
    sname = name;
    sargs = sargs;
  }

let cast_signatures fundefs =
  (List.map fundefs ~f:(fun (Sast.FunDef(n,args,(_,tret))) ->
    let decls = List.map args ~f:(fun (s,t)->(t,s)) in
    Cast.SigFunc(n,tret,decls)))

let cast_inclus incls =
  let defaults = ["iostream"; "string"; "vector"] in
  List.map defaults ~f:(fun s -> Cast.IncludeAngleBrack(s)) @
  List.map incls ~f:(fun s -> Cast.IncludeQuote(s))

let rec verify_no_assign_expr = function
  | Cast.DeclAssign(_,_) -> failwith "Cannot initialize a variable in this context"
  | Cast.Assign(_,_) -> failwith "Cannot assign to a variable in this context"
  | _ as expr -> verify_expr expr

and verify_expr = function
  (* traverse down a level of the ast *)
  | Cast.DeclAssign(_,expr) | Cast.Assign(_,expr) | Cast.Idx(_,expr)
  | Cast.Uniop(_,expr) ->
      verify_no_assign_expr expr
  | Cast.Binop(lexpr,_,rexpr) -> ignore(verify_no_assign_expr lexpr); verify_no_assign_expr rexpr
  | Cast.Call(callable,exprs) -> ignore(List.map exprs ~f:verify_no_assign_expr);
      begin match callable with
        | Cast.LambdaRefCap(_,_,stmts) -> ignore(List.map stmts ~f:verify_no_init_stmt)
        | Cast.Method(_) | Cast.Function(_,_,_) | Cast.Struct(_) -> ()
      end
  | Cast.LitUnit | Cast.LitBool(_) | Cast.LitInt(_) | Cast.LitFloat(_) | Cast.LitStr(_)
  | Cast.InitList(_) | Cast.Decl(_) | Cast.VarRef(_) | Noexpr -> ()

and verify_no_init_stmt = function
  | Cast.Block(stmts) -> ignore(List.map stmts ~f:verify_no_init_stmt)
  | Cast.Expr(expr) -> ignore(verify_expr expr)
  | Cast.Return(expr) -> ignore(verify_expr expr)
  | Cast.If(expr, lstmt, rstmt) ->
      ignore(verify_expr expr); ignore(verify_no_init_stmt lstmt);
      ignore(verify_no_init_stmt rstmt)
  | Cast.For(lexpr, mexpr, rexpr, stmt) ->
      ignore(verify_expr lexpr); ignore(verify_expr mexpr);
      ignore(verify_expr rexpr); ignore(verify_no_init_stmt stmt)
  | Cast.While(expr, stmt) -> ignore(verify_expr expr); ignore (verify_no_init_stmt stmt)
  | Cast.ForRange(_, expr, stmt) -> ignore(verify_expr expr); ignore(verify_no_init_stmt stmt)

let rec verify_no_init funs =
  ignore begin
    match funs with
      | [] -> ()
      | head::tail -> begin ignore(verify_no_init tail);
          ignore(
            let { fnamespace=_;fname=_;fargs=_;treturn=_; body=body } = head in
            List.map body ~f:verify_no_init_stmt
          )
        end
  end; funs

let strip_top_level = function
  | (Sast.Block(texprs),t) -> let (texprs,globals) = List.fold_left texprs ~init:([],[])
      (* change all top level inits to assignments *)
      ~f:(fun (texprs, globals) texpr ->
        match texpr with
          | Sast.Init(name,(expr,t),_),tunit ->
            (Sast.Assign([name],(expr,t)),tunit)::texprs, (t,name)::globals
          | _ -> texpr::texprs, globals
      )
      in ([Sast.Block(List.rev texprs),t], globals)
  | (Sast.LitUnit,Ast.Unit) as texprs -> [ texprs ], []
  | _ -> failwith "Internal Error: could not extract globals because top level was not Block"

let cast_of_sast (incls, fundefs, texpr, types) =
  let cast_incls = cast_inclus incls in
  let cast_fundefs = List.map fundefs ~f:(castfun_of_sastfun) in
  let cast_types = List.map types ~f:(casttype_of_sasttype) in
  let ssignatures = List.map cast_types
    ~f:(fun {sname=n; sargs=_} -> Cast.SigStruct(n)) in
  let fsignatures = cast_signatures fundefs in
  let (sexprs, globals) = strip_top_level texpr in
  let main_expr = (Sast.Block(sexprs @ [(Sast.LitInt(0),Ast.Int)]),Ast.Int) in
  let all_funs = castfun_of_sastfun (Sast.FunDef("main",[],main_expr))::cast_fundefs in
  let typedef_equality_funs = List.map types ~f:(equality_fun) in
  let typedef_nequality_funs = List.map types ~f:(nequality_fun) in
  let all_funs = typedef_equality_funs @ typedef_nequality_funs @ all_funs in
  let verified_funs = verify_no_init all_funs in
  cast_incls, ssignatures@fsignatures, globals, cast_types, verified_funs
```

9.2.24 src/interpret.ml

```
open Core.Std

open Sast

(* WIP -- created because peephole optimizations in optimize.ml had a lot of overlapping code w/ an interpreter *)

let (* rec *) interpreted_of_sast sexpr =
  let (expr, _) = sexpr in
  match expr with
  | Binop((LitBool(v1), t1), op, (LitBool(v2), _)) ->
      (match op with
      | Ast.Eq  -> Sast.LitBool(v1 = v2),  Ast.Bool
      | Ast.Neq -> Sast.LitBool(v1 <> v2), Ast.Bool
      | Ast.Lt  -> Sast.LitBool(v1 < v2),  Ast.Bool
      | Ast.Lte -> Sast.LitBool(v1 <= v2), Ast.Bool
      | _ ->
          failwith (sprintf "Internal error: can't use %s on type %s" (Ast.string_of_op op) (Ast.string_of_type t1))
      )

  | Binop((LitInt(v1), t1), op, (LitInt(v2), _)) ->
      (match op with
      | Ast.Add -> Sast.LitInt (v1 + v2),  Ast.Int
      | Ast.Sub -> Sast.LitInt (v1 - v2),  Ast.Int
      | Ast.Mul -> Sast.LitInt (v1 * v2),  Ast.Int
      | Ast.Div -> Sast.LitInt (v1 / v2),  Ast.Int
      | Ast.Mod -> Sast.LitInt (v1 % v2),  Ast.Int
      | Ast.Eq  -> Sast.LitBool(v1 = v2),  Ast.Bool
      | Ast.Neq -> Sast.LitBool(v1 <> v2), Ast.Bool
      | Ast.Lt  -> Sast.LitBool(v1 < v2),  Ast.Bool
      | Ast.Lte -> Sast.LitBool(v1 <= v2), Ast.Bool
```

```
        | _ ->
            failwith (sprintf "Internal error: can't use %s on type %s" (Ast.string_of_op op) (Ast.string_of_type t1))
        )

  | Binop((LitFloat(v1), t1), op, (LitFloat(v2), _)) ->
        (match op with
        | Ast.Add -> Sast.LitFloat(v1 +. v2), Ast.Float
        | Ast.Sub -> Sast.LitFloat(v1 -. v2), Ast.Float
        | Ast.Mul -> Sast.LitFloat(v1 *. v2), Ast.Float
        | Ast.Div -> Sast.LitFloat(v1 /. v2), Ast.Float
        | Ast.Eq  -> Sast.LitBool (v1 = v2),  Ast.Bool
        | Ast.Neq -> Sast.LitBool (v1 <> v2), Ast.Bool
        | Ast.Lt  -> Sast.LitBool (v1 < v2),  Ast.Bool
        | Ast.Lte -> Sast.LitBool (v1 <= v2), Ast.Bool
        | _ ->
            failwith (sprintf "Internal error: can't use %s. on type %s" (Ast.string_of_op op) (Ast.string_of_type t1))
        )

  | Binop((LitStr(v1), t1), op, (LitStr(v2), _)) ->
        (match op with
        | Ast.Concat -> Sast.LitStr (v1 ^ v2),  Ast.String
        | Ast.Eq     -> Sast.LitBool(v1 = v2),  Ast.Bool
        | Ast.Neq    -> Sast.LitBool(v1 <> v2), Ast.Bool
        | _ ->
            failwith (sprintf "Internal error: can't use %s. on type %s" (Ast.string_of_op op) (Ast.string_of_type t1))
        )

  | _ -> failwith "The interpreter doesn't support this kind of expression"
```

9.2.25 src/interpret.mli

```
open Core.Std

open Sast

val interpreted_of_sast : Sast.expr_typed -> Sast.expr_typed
```

9.2.26 src/log.ml

```
open Core.Std
open Printf

type log_level = Error | Warn | Info | Debug

(* For comparing logging levels *)
let int_of_level = function
  | Error -> 100
  | Warn -> 80
  | Info -> 60
  | Debug -> 40

let string_of_level = function
  | Error -> "error"
  | Warn -> "warning"
  | Info -> "info"
  | Debug -> "debug"

type color = Bold | Reset | Black | Red | Green | Yellow | Blue | Magenta | Cyan | White

let color_of_level = function
  | Error -> Red
  | Warn -> Yellow
  | Info -> Blue
  | Debug -> Cyan

let string_of_color color =
  let escape_of_color = function
    | Reset   ->  0
    | Bold    ->  1
    | Black   -> 30
    | Red     -> 31
    | Green   -> 32
    | Yellow  -> 33
    | Blue    -> 34
    | Magenta -> 35
    | Cyan    -> 36
    | White   -> 37
  in
  sprintf "\027[%dm" (escape_of_color color)

let min_level = ref Warn

let set_min_level l =
  min_level := l

let print level fmt =
  let prefix = (string_of_color (color_of_level level)) ^ (string_of_color Bold) ^
    (string_of_level level) ^ ":" ^ (string_of_color Reset) in
  let printer = if int_of_level level >= int_of_level !min_level then fprintf else ifprintf in
  printer stderr ("%s " ^^ fmt ^^ "\n%!") prefix

let error fmt = print Error fmt
let warn  fmt = print Warn  fmt
let info  fmt = print Info  fmt
let debug fmt = print Debug fmt
```

9.2.27 src/log.mli

```
open Core.Std

type log_level =
  | Error (* Compile error *)
  | Warn  (* Compile warning *)
  | Info  (* Informational message (-v) *)
  | Debug (* Low-level information (-vv) *)

val string_of_level : log_level -> string

type color = Bold | Reset | Black | Red | Green | Yellow | Blue | Magenta | Cyan | White

val set_min_level : log_level -> unit

val error : ('a, out_channel, unit) format -> 'a
val warn  : ('a, out_channel, unit) format -> 'a
val info  : ('a, out_channel, unit) format -> 'a
val debug : ('a, out_channel, unit) format -> 'a
```

9.2.28 src/Makefile

```
OCAMLBUILD = corebuild
CFLAGS = -safe-string
BUILDFLAGS = -use-ocamlfind -cflags $(CFLAGS)

.PHONY: nhc
nhc: update-version
	$(OCAMLBUILD) $(BUILDFLAGS) nhc.native

.PHONY: update-version
update-version:
	./update_version.sh

.PHONY: test
test: nhc
	$(OCAMLBUILD) $(BUILDFLAGS) tester.native
	./tester.native ../test

.PHONY: clean
clean:
	$(OCAMLBUILD) -clean
	rm -f version.ml
```

9.2.29 src/nhc.ml

```
open Core.Std

open Ast
open Cast
open Log
open Noincl_ast
open Optimize_manager
open Typed_ast
open Cpp_sast
open Version

(* string -> Ast -> Noincl_ast -> Typed_ast -> Optimize_manager -> Cast *)
let do_compile src_path bin_path keep_ast keep_il =
  let ast = Noincl_ast.ast_of_filename src_path in
  if keep_ast then
    let ast_path = bin_path ^ ".ast"
    and (fdefs, externs, exprs, tdefs) = ast in
    Out_channel.write_all ast_path ~data:(string_of_prog_struc ([], fdefs, externs, exprs, tdefs))
  else ();
  let (incls, funs, sast, tdefaults) = sast_of_ast ast in
  let sast = optimized_of_sast sast in
  let cast = cast_of_sast (incls, funs, sast, tdefaults) in
  let cpp = string_of_program cast in
  if keep_il then
    let il_path = bin_path ^ ".cpp" in
    Out_channel.write_all il_path ~data:cpp
  else ();
  let cxx_opts = "-Wall -pedantic -fsanitize=address -std=c++14 -O2" in
  let cxx_incls = "-I/usr/local/include/ -L/usr/local/lib/ -lstk" in
  let cxx = sprintf "clang++ %s %s -xc++ - support.cpp" cxx_opts cxx_incls in
  Log.info "Calling cxx with opts: %s" cxx;
  let ch = Unix.open_process_out cxx in
  Out_channel.output_string ch cpp;
  if Unix.close_process_out ch <> Result.Ok( () ) then
    failwith "Internal error: C++ compilation failed"
  else ();
  ()

let command =
  Command.basic
    ~summary:"Compiler for the ♫ #ïÿŔâČč  language"
    ~readme:(fun () -> "For more information, visit https://github.com/el2724/note-hashtag")
    Command.Spec.(
      empty
      +> flag "-A" no_arg ~doc:" output internal representation (syntax tree)"
      +> flag "-c" (optional string) ~doc:"file.nh compile the specified file"
      +> flag "-o" (optional_with_default "a.out" string) ~doc:"file write output to the specified file"
      +> flag "-S" no_arg ~doc:" output intermediate language representation (C++)"
      +> flag "-v" no_arg ~doc:" print verbose debugging information"
      +> flag "-vv" no_arg ~doc:" print extra verbose debugging information"
    )
    ( (* Handler *)
      fun show_ast infile_path outfile_path show_il verbose1 verbose2 () ->
        let log_level = if verbose2 then Debug else if verbose1 then Info else Warn in
        Log.set_min_level log_level;
        Log.debug "Parsed command line options:";
        Log.debug "  minimum log_level: %s" (string_of_level log_level);
        Log.debug "  compiling file (empty for stdin): %s" (match infile_path with None -> "" | Some(s) -> s);
        Log.debug "  writing executable to: %s" outfile_path;
        Log.debug "  keep intermediate representation: %B" show_il;
        Log.debug "  keep syntax tree: %B" show_ast;
        try
          do_compile infile_path outfile_path show_ast show_il;
        (* Catch any unhandled exceptions to suppress the nasty-looking message *)
        with
        | Scanner.Lexing_error(msg) -> Log.error "Lexing error: %s" msg; exit 1
        | Parsing.Parse_error -> Log.error "Syntax error"; exit 2
        | Failure(msg) | Sys_error(msg) ->
            Log.error "%s" msg; Log.debug "call stack:\n%s" (Printexc.get_backtrace ()); exit 3
    )

let _ =
  try Command.run ~version:(Version.release ()) ~build_info:(Version.build ()) command;
  with Sys_error(msg) -> Log.error "Argument error: %s" msg; exit 4
```

9.2.30 src/noincl_ast.ml

```
open Core.Std

type program = Ast.fundef list * Ast.externfun list * Ast.expr list * Ast.typedef list

let ast_of_inchan inchan = Parser.program Scanner.token (Lexing.from_channel inchan)

let rec noinclu incls_ref pwd (incls, fdefs, externs, exprs, tdefs) =
  (* The user just types "include myfile" so we need to prepend the search directory and append the extension *)
  let incls = List.map incls ~f:(fun incl -> Filename.concat pwd (incl ^ ".nh")) in
  (* Grab ASTs for all the includes *)
  let incl_fdefs, incl_externs, incl_exprs, incl_tdefs = List.fold_left incls ~init:([], [], [], [])
    ~f:(fun (fdefs, externs, exprs, tdefs) next_incl ->
      let next_fdefs, next_externs, next_exprs, next_tdefs =
        (* for each file, ast recursively *)
        if List.mem !incls_ref next_incl then
          [], [], [], []
        else
          let inchan = In_channel.create next_incl in
          let next_ast = ast_of_inchan inchan in
          begin
            let realpath = Filename.realpath next_incl in
            incls_ref := realpath :: !incls_ref;
            noinclu incls_ref (Filename.dirname realpath) next_ast
          end
      in
      (fdefs @ next_fdefs, externs @ next_externs, exprs @ next_exprs, tdefs @ next_tdefs)
    )
  in
  (* Prepend the included ASTs to our AST *)
  incl_fdefs @ fdefs, incl_externs @ externs, incl_exprs @ exprs, incl_tdefs @ tdefs

let ast_of_filename name =
  (* toplevel ast *)
  let get_inchan = function
    | None -> In_channel.stdin
    | Some(filename) -> In_channel.create filename
  in
  let (incls, fdefs, externs, exprs, tdefs) = ast_of_inchan (get_inchan name) in
  try
    (* keep track of already included files *)
    let seen_incls = ref (match name with None -> [] | Some(x) -> [ Filename.realpath x ]) in
    let pwd = match name with None -> "./" | Some(x) -> Filename.dirname x in
    noinclu seen_incls pwd ("../lib/std" :: incls, fdefs, externs, exprs, tdefs)
  with Unix.Unix_error(error, _, fname) -> failwith (sprintf "%s: %s" fname (Unix.error_message error))
```

9.2.31 src/noincl_ast.mli

```
open Core.Std

type program = Ast.fundef list * Ast.externfun list * Ast.expr list * Ast.typedef list

val ast_of_filename : string option -> program
```

9.2.32 src/optimize.ml

```
open Core.Std

open Sast

let rec constfold sexpr =
  let (expr, exprt) = sexpr in
  match expr with
  | Binop(lexpr, op, rexpr) ->
      let lexpr = constfold lexpr and rexpr = constfold rexpr in
      let (v1, _) = lexpr and (v2, _) = rexpr in
      let foldable =
        match v1, v2 with
        | LitInt(v1), LitInt(v2) -> Log.debug "constfold found: %d %s %d" v1 (Ast.string_of_op op) v2; true
        | LitFloat(v1), LitFloat(v2) -> Log.debug "constfold found: %f %s. %f" v1 (Ast.string_of_op op) v2; true
        | LitStr(v1), LitStr(v2) -> Log.debug "constfold found: \"%s\" %s \"%s\"" v1 (Ast.string_of_op op) v2; true
        | _ -> false
      in
      if foldable then Interpret.interpreted_of_sast (Binop(lexpr, op, rexpr), exprt)
      else Binop(lexpr, op, rexpr), exprt

  (* Now we get to write out all the recursive cases...there's gotta be a better way of doing this *)

  | Uniop(op, expr) ->
      let expr = constfold expr in
      Uniop(op, expr), exprt

  | FunApply(fname, exprs) ->
      let exprs = List.map exprs ~f:constfold in
      FunApply(fname, exprs), exprt

  | ArrIdx(varref, expr) ->
      let expr = constfold expr in
      ArrIdx(varref, expr), exprt

  | Arr(exprs, t) ->
      let exprs = List.map exprs ~f:constfold in
      Arr(exprs, t), exprt

  | Block(exprs) ->
      let exprs = List.map exprs ~f:constfold in
      Block(exprs), exprt

  | Conditional(e1, e2, e3) ->
      let e1 = constfold e1
      and e2 = constfold e2
      and e3 = constfold e3 in
      Conditional(e1, e2, e3), exprt

  | For(vdecl, e1, e2) ->
      let e1 = constfold e1
      and e2 = constfold e2 in
      For(vdecl, e1, e2), exprt

  | Init(name, expr, mutability) ->
      let expr = constfold expr in
      Init(name, expr, mutability), exprt

  | Assign(name, expr) ->
      let expr = constfold expr in
      Assign(name, expr), exprt

  | Struct(name, defaults) ->
      let defaults = List.map defaults ~f:(fun (name, expr) -> (name, constfold expr)) in
      Struct(name, defaults), exprt

  (* If we don't have children to check or an optimization to apply, just return the same thing *)
  | _ -> sexpr
```

9.2.33 src/optimize.mli

```
open Core.Std

open Sast

val constfold : Sast.expr_typed -> Sast.expr_typed
```

9.2.34 src/optimize_manager.ml

```
open Core.Std

open Optimize

let passes = [ Optimize.constfold ]

let optimized_of_sast sexpr =
  List.fold_left passes ~init:sexpr ~f:(fun sexpr pass -> pass sexpr)
```

9.2.35 src/optimize_manager.mli

```
open Core.Std

open Sast

val optimized_of_sast : Sast.expr_typed -> Sast.expr_typed
```

9.2.36 src/parser.mly

```
%{
  open Core.Std

  open Ast
%}

```

```
%token SEP
%token LPAREN RPAREN BRACKS BRACES LBRACK RBRACK LBRACE RBRACE DOT_LPAREN CONCAT COMMA
%token PLUS MINUS TIMES DIVIDE MOD
%token EQ NEQ LT LTE GT GTE
%token NOT AND OR
%token SHARP FLAT COLON OCTAVE
/* Note: "a = b = 3" is valid; 3 is assigned to b, and the value of that */
/* expression is assigned to a. */
%token ASSIGN CONST
%token TILDE
%token IF THEN ELSE BE UNLESS INWHICHCASE FOR IN DO
%token TYPE
%token BLING
%token EOF
%token INCLUDE FUN EXTERN LARROW RARROW
%token THROW
%token INIT

%token <string> ID_LOWER
%token <string> ID_UPPER

%token <bool> LIT_BOOL
%token <int> LIT_INT
%token <float> LIT_FLOAT
%token <string> LIT_STR
%token TYPE_UNIT TYPE_BOOL TYPE_INT TYPE_FLOAT TYPE_STR

%nonassoc ELSE INWHICHCASE DO
%left SEP
%nonassoc ASSIGN
%left OR
%left AND
%left EQ NEQ
/* x < y < z can never be valid because can't use < on bool type. */
%nonassoc LT LTE GT GTE
%left CONCAT
%left COLON
```

```
%left COMMA
%left OCTAVE
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left BLING

/* Unary Operators */
%nonassoc NOT
%right prec_unary_minus
%left SHARP FLAT

%start program
%type <Ast.program> program

%%

program:
| sep_star EOF { [], [], [], [], [] }
| sep_star program_header_follow_body program_body EOF
  { (fun (fdefs, externs, exprs, structdefs) -> ($2, fdefs, externs, exprs, structdefs)) $3 }
| sep_star program_body EOF { (fun (fdefs, externs, exprs, structdefs) -> ([], fdefs, externs, exprs, structdefs)) $2 }
| sep_star program_header EOF { $2, [], [] ,[], [] }

program_header_follow_body:
| INCLUDE ID_LOWER include_list sep_plus { $2 :: List.rev $3 }

program_header:
| INCLUDE ID_LOWER include_list sep_star { $2 :: List.rev $3 }

include_list:
| /* nothing */ { [] }
| include_list sep_plus INCLUDE ID_LOWER { $4 :: $1 }

program_body:
| struct_declaration program_body_list sep_star
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, externs, exprs, $1 :: structdefs)) $2 }
| fun_def program_body_list sep_star
```

```
  { (fun (fdefs, externs, exprs, structdefs) -> ($1 :: fdefs, externs, exprs, structdefs)) $2 }
| extern_fun program_body_list sep_star
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, $1 :: externs, exprs, structdefs)) $2 }
| expr program_body_list sep_star
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, externs, $1 :: exprs, structdefs)) $2 }

program_body_list:
| /* nothing */ { [], [], [], [] }
| program_body_list sep_plus struct_declaration
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, externs, exprs, structdefs @ [ $3 ])) $1 }
| program_body_list sep_plus fun_def
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs @ [ $3 ], externs, exprs, structdefs)) $1 }
| program_body_list sep_plus extern_fun
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, externs @ [ $3 ], exprs, structdefs)) $1 }
| program_body_list sep_plus expr
  { (fun (fdefs, externs, exprs, structdefs) -> (fdefs, externs, exprs @ [ $3 ], structdefs)) $1 }

struct_declaration:
| TYPE ID_LOWER ASSIGN LBRACE sep_star asn_list sep_star RBRACE { TypeDef($2, List.rev $6) }

fun_def:
| FUN ID_UPPER ID_LOWER_list ASSIGN expr { FunDef($2, $3, $5) }

extern_fun:
| EXTERN LIT_STR LIT_STR LIT_STR FUN ID_UPPER typename_list RARROW typename { ExternFunDecl($2, $3, $4, $6, $7, $9) }

typename_list:
| /* nothing */ { [] }
| typename typename_list { $1 :: $2 }

typename:
| TYPE_UNIT  { Unit }
| TYPE_BOOL  { Bool }
| TYPE_INT   { Int }
| TYPE_FLOAT { Float }
| TYPE_STR   { String }
| ID_LOWER   { Type($1) }
| typename BRACES { Array($1) }

block:
| sep_list sep_star { Block(List.rev $1) }
| expr sep_list sep_star { Block($1 :: List.rev $2) }

sep_list:
| /* nothing */ { [] }
| sep_list sep_plus expr { $3 :: $1 }

/* Helper: One or more separators */
sep_plus:
| SEP { () }
| SEP sep_plus { () }

/* Helper: Zero or more separators */
sep_star:
| /* nothing */ { () }
| SEP sep_star { () }

expr:
| apply     { $1 }
| non_apply { $1 }
| arith     { $1 }
| bool      { $1 }
| OCTAVE non_apply non_apply {Binop($3, Octave, $2)}
| expr COLON expr {Binop($1, Zip, $3)}
| asn_toplevel { $1 }
| expr CONCAT expr { Binop($1, Concat, $3) }
| control { $1 }
| THROW non_apply { Throw($2) }

control:
| IF sep_expr_sep THEN sep_expr_sep ELSE sep_star expr { Conditional($2,$4,$7) }
| BE sep_expr_sep UNLESS sep_expr_sep INWHICHCASE sep_star expr { Conditional($4,$7,$2) }
| FOR sep_star ID_LOWER sep_star IN sep_expr_sep DO sep_star expr { For($3,$6,$9) }
| INIT ID_LOWER { StructInit($2, []) }
| INIT ID_LOWER LBRACE sep_star asn_list sep_star RBRACE { StructInit($2, List.rev $5) }

ID_LOWER_list:
| /* nothing */ { [] }
| ID_LOWER ID_LOWER_list { $1 :: $2 }

stmt_list_plus:
| non_apply { [$1] }
| stmt_list_plus non_apply { $2 :: $1 }

apply:
| ID_UPPER args_list { FunApply($1, $2) }

args_list:
| /* nothing */       { [] }
| non_apply args_list { $1 :: $2 }

non_apply:
| var_ref { VarRef($1) }
| var_ref DOT_LPAREN expr RPAREN { ArrIdx($1, $3) }
| LPAREN block RPAREN { $2 } /* we get unit () notation for free (see block) */
| lit { $1 }
| non_apply OCTAVE non_apply { Binop($1, Octave, $3) }
| non_apply COMMA non_apply {Binop($1, Chord, $3)}
| music     { $1 }

sep_expr_sep:
| sep_star expr sep_star { $2 }

lit:
| LIT_BOOL         { LitBool($1) }
| LIT_INT          { LitInt($1) }
| LIT_FLOAT        { LitFloat($1) }
| LIT_STR          { LitStr($1) }
| TILDE            { StructInit("chord", []) }
| lit_array        { $1 }

```

```
lit_array:
| LBRACE stmt_list_plus RBRACE { Arr((List.rev $2), None) }
| LBRACK stmt_list_plus RBRACK { ArrMusic((List.rev $2)) }
| typename   BRACES  { Arr([], Some($1)) }

arith:
| MINUS expr %prec prec_unary_minus { Uniop(Neg, $2) }
| expr PLUS   expr { Binop($1, Add, $3) }
| expr MINUS  expr { Binop($1, Sub, $3) }
| expr TIMES  expr { Binop($1, Mul, $3) }
| expr DIVIDE expr { Binop($1, Div, $3) }
| expr MOD    expr { Binop($1, Mod, $3) }

bool:
| cmp { $1 }
| logic { $1 }

cmp:
| expr EQ  expr { Binop($1, Eq,  $3) }
| expr NEQ expr { Binop($1, Neq, $3) }
| expr LT  expr { Binop($1, Lt,  $3) }
| expr LTE expr { Binop($1, Lte, $3) }
| expr GT  expr { Binop($3, Lt,  $1) }
| expr GTE expr { Binop($3, Lte, $1) }

logic:
|      NOT expr { Uniop(Not, $2) }
| expr AND expr { Binop($1, And, $3) }
| expr OR  expr { Binop($1, Or,  $3) }

music:
| non_apply FLAT     { Uniop(Flat, $1)}
| non_apply SHARP    { Uniop(Sharp, $1)}

var_ref:
| ID_LOWER { [ $1 ] }
| ID_LOWER BLING var_ref { $1 :: $3 }
```

```

asn_toplevel:
| asn { $1 }
/* ID_LOWER because you can't do `const a$b = 10` */
| CONST ID_LOWER ASSIGN expr { Assign([ $2 ], $4, Immutable) }

asn:
| var_ref ASSIGN expr { Assign($1, $3, Mutable) }

asn_list:
| asn { [ $1 ] }
| asn_list sep_plus asn { $3 :: $1 }
```

9.2.37 src/sast.ml

```
open Core.Std

open Ast

type variable_name = Ast.var_reference
type new_variable_name = string
type function_name =
  | NhFunction of string
  (* Header file name, namespace, C++ function name *)
  | CppFunction of string * string * string

type expr_detail =
  | LitBool of bool
  | LitInt of int
  | LitFloat of float
  | LitStr of string
  | LitUnit
  | Binop of expr_typed * Ast.binary_operator * expr_typed
  | Uniop of Ast.unary_operator * expr_typed
  | VarRef of variable_name
  | FunApply of function_name * expr_typed list
  | ArrIdx of variable_name * expr_typed
  | Arr of (expr_typed list) * Ast.t
  | Block of expr_typed list
  | Conditional of expr_typed * expr_typed * expr_typed
  | For of (new_variable_name * Ast.t) * expr_typed * expr_typed
  | Exit of int
  | Init of new_variable_name * expr_typed * Ast.mutability
  | Assign of variable_name * expr_typed
  | Struct of type_name * ((string * expr_typed) list)

and expr_typed = expr_detail * Ast.t

(* type name, fields and default values *)
type tdefault = TDefault of type_name * ((string * expr_typed) list)

type fundef_typed = FunDef of string * ((string * Ast.t) list) * expr_typed

(* C++ includes, function declarations, expressions, types (with defaults) *)
type program_typed = string list * fundef_typed list * expr_typed list * tdefault list
```

9.2.38 src/scanner.mll

```
{
  open Core.Std

  open Parser

  exception Lexing_error of string
}

let lowercase = ['a'-'z']
let uppercase = ['A'-'Z']
let letter = lowercase | uppercase
let digit = ['0'-'9']
let newline = ('\n' | '\r' | "\r\n")
let whitespace = [' ' '\t']
let separator = (newline | ';')

(* Used for float parsing *)
let hasint = digit+ '.' digit*
let hasfrac = digit* '.' digit+
let hasexp = 'e' ('+'? | '-') digit+

rule token = parse
| '\\' newline { token lexbuf }
| separator { SEP }
| whitespace { token lexbuf }
| eof { EOF }
(* Scoping *)
| '(' { LPAREN }
| ')' { RPAREN }
| '{' (whitespace | newline)* '}'{ BRACES }
| '[' { LBRACK }
| ']' { RBRACK }
| '{' { LBRACE }
| '}' { RBRACE }
(* Operators *)
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| '%' { MOD }
| '=' { ASSIGN }
| "==" { EQ }
| "!=" { NEQ }
| '<' { LT }
| "<=" { LTE }
| '>' { GT }
| ">=" { GTE }
| ".(" { DOT_LPAREN }
| '.' { CONCAT }
| ',' {COMMA}
| '!' { NOT }
| "&&" { AND }
| "||" { OR }
| '#' {SHARP}
| 'b' {FLAT}
| ':' {COLON}
| '@' {OCTAVE}
| '~' { TILDE }
| "->" { RARROW }
| "<-" { LARROW }
| '$' { BLING }
(* Keywords *)
(* Regex conflicts are resolved by order! Place all keywords in this section or ID_LOWER will eat them up. *)
| "unit" { TYPE_UNIT }
| "bool" { TYPE_BOOL }
| "int" { TYPE_INT }
| "float" { TYPE_FLOAT }
| "string" { TYPE_STR }
| "true" { LIT_BOOL(true) }
| "false" { LIT_BOOL(false) }
| "fun" { FUN }
| "include" { INCLUDE }
| "if" { IF }
| "then" { THEN }
| "else"{ ELSE }
| "be" { BE }
| "unless" { UNLESS }
| "inwhichcase" { INWHICHCASE }
| "for" { FOR }
| "in" { IN }
| "do" { DO }
| "throw" { THROW }
| "type" { TYPE }
| "init" | "beget" | "bringintobeing" { INIT }
| "extern" { EXTERN }
| "const" { CONST }
(* Literals *)
| digit+ as lit { LIT_INT(Int.of_string lit) }
| ((hasint | hasfrac) hasexp?) | (digit+ hasexp) as lit { LIT_FLOAT(Float.of_string lit) }
| '"' (('\\' '"'| [^'"'])* as str) '"' { LIT_STR(Scanf.unescaped str) }
(* Identifiers *)
| (lowercase | '_') (letter | digit | '_')* as lit { ID_LOWER(lit) }
| uppercase (letter | digit | '_')* as lit { ID_UPPER(lit) }
(* Comments *)
| "//" { comment_oneline lexbuf }
| "/*" { comment_multiline 0 lexbuf }
| "*/" { raise (Lexing_error("You have a '*/' without a matching '/*'")) }
| _ as c { raise (Lexing_error("Unknown token '" ^ Char.to_string c ^ "'")) }

and comment_oneline = parse
| (newline | eof) { SEP } (* End of single line comment *)
| _ { comment_oneline lexbuf }

and comment_multiline depth = parse
| "/*" { comment_multiline (depth + 1) lexbuf }
| "*/" { if depth = 0 then token lexbuf else comment_multiline (depth - 1) lexbuf }
| eof { raise (Lexing_error("You have a '/*' without a matching '*/'")) }
| _ { comment_multiline depth lexbuf }
```

9.2.39 src/support.cpp

```
#include "support.hpp"

#include <stk/Wurley.h>
#include <stk/Instrmnt.h>
#include <stk/FileLoop.h>
#include <stk/FileWvOut.h>

const unit_t LIT_UNIT = 0;

std::default_random_engine nh_support::myrand(static_cast<unsigned int>(::time(0)));

unit_t nh_support::print_string(std::string output)
{
  std::cout << output;
  return LIT_UNIT;
}

int64_t nh_support::int_of_float(double n)
{
  return static_cast<int64_t>(n);
}

double nh_support::float_of_int(int64_t n)
{
  return static_cast<double>(n);
}

template <typename T>
void safe_insert(std::vector<T> &v, size_t idx, T obj)
{
  while (v.size() <= idx) v.push_back(T());
  v[idx] = obj;
}

template <typename T>
T safe_at(std::vector<T> &v, size_t idx)
{
  if (v.size() <= idx) return T();
  else return v[idx];
}

unit_t nh_support::render_impl(
  std::vector<std::vector<std::vector<double>>> frequencies,
  std::vector<std::vector<double>> durations,
  std::vector<double> volumes,
  std::string filename)
{
  auto instrument_allocator = []() { return stk::Wurley(); };
  const double sample_rate = 44100.0;
  // Set the global sample rate before creating class instances.
  stk::Stk::setSampleRate(sample_rate);
  stk::Stk::setRawwavePath("../stk/rawwaves/");

  auto instrument = instrument_allocator();

  std::vector<double> samples;
```

```cpp
// Loop through tracks
for (size_t i_track = 0; i_track < frequencies.size(); i_track++) {
  std::vector<std::vector<double>> track = frequencies[i_track];

  const double volume_scale = volumes[i_track] / volumes.size();
  size_t i = 0; // Index for writing into samples array
  // Loop through chords
  for (size_t i_chord = 0; i_chord < track.size(); i_chord++) {
    std::vector<double> chord = track[i_chord];

    const double samples_total = sample_rate * durations[i_track][i_chord];
    const size_t samples_sound = static_cast<size_t>(0.9 * samples_total);
    // Loop through notes
    for (double note : chord) {
      // Rests have a frequency <= 0.0
      bool is_silence = (note <= 0.0);
      if (!is_silence) instrument.noteOn(note, 1.0);
      // Offset i + calculate number of samples to output
      const size_t end = i + samples_sound;
      for (size_t j = i; j < end; j++) {
        // Need to normalize so chords are all the same volume
        auto sample = safe_at(samples, j) + (is_silence ? 0.0 : volume_scale * instrument.tick() / chord.size());
        safe_insert(samples, j, sample);
      }
    }
    // Done outputting this note, so increment our index to the output buffer
    // Since we output audio samples until samples_sound, incrementing by samples_total gives us some silence at the
    // end of each note.
    i += static_cast<size_t>(samples_total);
  }
}

// Open a 16-bit, one-channel WAV formatted output file
stk::Stk::showWarnings(false); // Suppress the annoying "creating WAV file" message
stk::FileWvOut output(filename, 1, stk::FileWrite::FILE_WAV, stk::Stk::STK_SINT16);
stk::Stk::showWarnings(true);
for (double sample : samples) {
    output.tick(sample);
  }
  output.closeFile();

  return LIT_UNIT;
}
```

9.2.40 src/support.hpp

```
#ifndef __support_hpp__
#define __support_hpp__

#include <algorithm>
#include <cmath>
#include <iostream>
#include <random>
#include <string>
#include <vector>

typedef int unit_t;
extern const unit_t LIT_UNIT;

namespace nh_support
{
  extern std::default_random_engine myrand;

  unit_t print_string(std::string output);

  int64_t int_of_float(double n);
  double float_of_int(int64_t n);

  template<typename T>
  T concat(T v1, T v2)
  {
    T result(v1);
    std::copy(v2.begin(), v2.end(), std::back_inserter(result));
    return result;
  }
```

```cpp

  template<typename T>
  std::vector<T> shuffle(std::vector<T> v)
  {
    std::shuffle(v.begin(), v.end(), myrand);
    return v;
  }

  unit_t render_impl(
    std::vector<std::vector<std::vector<double>>> frequencies,
    std::vector<std::vector<double>> durations,
    std::vector<double> volumes,
    std::string filename
  );
}

#endif
```

9.2.41 src/tester.ml

```ocaml
open Core.Std
open Filename
open Printf
open Unix

open Log

type filesystem = File of string | Directory of filesystem list

let rec read_out ch l =
  try
    let l = l ^ input_line ch in read_out ch l
  with End_of_file ->
    ignore (In_channel.close ch); l

let test_file file =
  let retval_pass = Result.Ok( () ) in
  (* Compile and check return value *)
  let nhc = Unix.open_process_in ("./nhc.native -c " ^ file) in
  print_string (In_channel.input_all nhc); (* No good way to do this -- just have to redirect stdout manually *)
  let nhc_retval = Unix.close_process_in nhc in
  if nhc_retval <> retval_pass then false else
  (* Run executable and check return value *)
  let child = Unix.open_process_in "./a.out" in
  let output = In_channel.input_all child
  and child_retval = Unix.close_process_in child in
  if child_retval <> retval_pass then false else
  (* Check child's output *)
  let expected = In_channel.read_all (Filename.chop_extension file ^ ".out") in
  output = expected

let test_files files =
  let print_test_file path =
    let passed = test_file path in
    (if passed then Log.debug "âIJĚ  %s" else Log.debug "ð§Ťě  %s") (Filename.basename path);
    passed
  in
  List.map files ~f:print_test_file

let rec filenames_of_filesystem fs =
  match fs with
  | File(filename) -> if Filename.check_suffix filename ".nh" then [ filename ] else []
  | Directory(fs_list) -> List.fold_left (List.map fs_list ~f:filenames_of_filesystem) ~init:[] ~f:(@)

let readdir_no_ex dirh =
  try
    Some (readdir dirh)
  with
    End_of_file -> None

let rec read_directory path =
  let dirh = opendir path in
  let rec loop () =
    let filename = readdir_no_ex dirh in
    match filename with
    | None -> []
    | Some "." -> loop ()
    | Some ".." -> loop ()
    | Some filename ->
      let pathname = path ^ "/" ^ filename in
      let stat = lstat pathname in
      let this = if stat.st_kind = S_DIR then read_directory pathname else File pathname in
      this :: loop () in
      Directory(loop ());;

let _ =
  Log.set_min_level Debug;
  let path = Sys.argv.(1) in
  let fs = read_directory path in
  let tests = filenames_of_filesystem fs in
  let results = test_files tests in
  let num_passed = List.count results ~f:(fun pass -> pass) in
  Log.info "Passed %d of %d tests" num_passed (List.length results)
```

9.2.42 src/typed_ast.ml

```

open Core.Std
open Sast

exception Cant_infer_type of string

(* environment *)
type symbol_table = {
  parent: symbol_table option;
  variables: (string * Ast.t * Ast.mutability) list;
}

type environment = {
  scope: symbol_table; (* vars symbol table *)
  functions: (int * Ast.fundef) list; (* (num args * fundef) list *)
  extern_functions: Ast.externfun list;
  types: Sast.tdefault list;
}

let rec get_top_scope scope = match scope.parent with
  | None -> scope
  | Some(scope) -> get_top_scope scope

let rec has_cycle_rec nodes (self, children) seen =
  match List.find !seen ~f:(fun n -> n = self) with
    | Some(_) -> true
    | None -> seen := self :: !seen; List.fold_left
        (List.map children
          ~f:(fun child ->
            match List.find nodes ~f:(fun (t,_) -> t = child) with
              | None -> has_cycle (List.filter nodes ~f:(fun (n,_) -> not (List.mem !seen n)))
              | Some(head) -> has_cycle_rec nodes head seen
          )
        )
        ~init:(false) ~f:(||)

and has_cycle nodes = match nodes with
  | [] -> false
  | head::_ -> has_cycle_rec nodes head (ref [])

let rec find_field types t access_list =
  (* make sure we are accessing a Type() *)
  let tname = match t with
    | Ast.Type(tname) -> tname
    | _ -> failwith ("cannot field access type "^Ast.string_of_type t)
  in
  (* make sure type exists *)
  let TDefault(_, fields) =
    match List.find types ~f:(fun (TDefault(s,_)) -> s = tname) with
      | None -> failwith ("Internal Error: Could not find type "^tname)
      | Some(x) -> x
  in match access_list with
    | [] -> failwith "Internal Error: tried field access with empty list"
    | field ::tail -> let (n,(x,t)) = match List.find fields ~f:(fun (s,_) -> s = field) with
```

```
                           | None -> failwith ("field "^field^" not found in type "^tname)
                           | Some(x) -> x
                         in match tail with
                           | [] -> (n,(x,t))
                           | _ -> find_field types t tail



let replace_add_fun l item =
  let (Sast.FunDef(name, tparams, (_, _))) = item in
  l := item :: List.filter !l
    ~f:(fun (Sast.FunDef(n, tps, (_, _))) -> name <> n || tparams <> tps)

let rec find_variable (scope: symbol_table) name =
  match List.find scope.variables ~f:(fun (s, _, _) -> s = name) with
  | None -> begin
    match scope.parent with
      | Some(parent) -> find_variable parent name
      | None -> raise Not_found
    end
  | Some(x) -> x

let find_ref_type env name fields =
  (* Find the variable called name *)
  let (_, t, _) = find_variable env.scope name in
  (* Call find_field if we're accessing a field of a user-defined type *)
  if fields <> [] then (let (_, (_, t)) = find_field env.types t fields in t) else t

let find_seen_function functions name tparams =
  List.find functions ~f:(fun (n,tps) -> name=n && tps=tparams)

let find_function functions name num_args = match
  List.find functions ~f:(function (n, Ast.FunDef(fname,_,_)) -> name = fname && num_args = n)
  with
    | Some(x) -> Log.debug "Found %s as nh function" name; x
    | None ->
      Log.debug "Couldn't find %s in nh functions" name;
```

```
   failwith ("Function " ^ name ^ " can’t be called with these arguments")

let find_extern externs name arg_types =
 match List.find externs
  ~f:(fun (Ast.ExternFunDecl(_, _, _, fname, ftypes, _)) -> fname = name && ftypes = arg_types) with
 | Some(x) -> Log.debug "Found %s as extern function" name; x
 | None ->
  Log.debug "Couldn’t find %s as extern function" name;
  failwith ("Function " ^ name ^ " can’t be called with these arguments")

let is_nh_function env name =
 List.exists env.functions ~f:(fun (_, Ast.FunDef(fname, _, _)) -> fname = name)

let is_cpp_function env name =
 List.exists env.extern_functions ~f:(fun (Ast.ExternFunDecl(_, _, _, fname, _, _)) -> fname = name)

(* Make sure functions are unique with each other and externs *)
let check_unique_functions fundefs externs =
 List.fold_left fundefs ~init:[]
 ~f:(fun defs astfundef ->
  let Ast.FunDef(fname, args, expr) = astfundef in
  let nfundef = (List.length args, Ast.FunDef(fname, args, expr)) in
  if List.exists externs
   ~f:(fun (Ast.ExternFunDecl(_, _, _, nh_name, arg_types, _)) ->
    nh_name = fname && List.length arg_types = List.length args)
  then failwith ("Function " ^ fname ^ " is already declared as an external function")
  else
   if List.mem defs nfundef
   then failwith ("Function " ^ fname ^ " is already defined")
   else nfundef :: defs
 )

let chord_of expr t =
 begin match t with
  (* use function in standard library on chordable (chord, pitch, int) exprs *)
  | Ast.Int -> Ast.FunApply("ChordOfPitch", [Ast.FunApply("PitchOfInt", [expr])])
  | Ast.Type("pitch") -> Ast.FunApply("ChordOfPitch", [expr])
```

```
    | Ast.Type("chord") -> expr
    | _ -> failwith "This expression is not chordable"
  end

let rec sast_expr ?(seen_funs = []) ?(force = false) env tfuns_ref e =
  let sast_expr_env = sast_expr ~seen_funs:seen_funs ~force:force env tfuns_ref in
  match e with
  | Ast.LitBool(x) -> Sast.LitBool(x), Ast.Bool
  | Ast.LitInt(x) -> Sast.LitInt(x), Ast.Int
  | Ast.LitFloat(x) -> Sast.LitFloat(x), Ast.Float
  | Ast.LitStr(x) -> Sast.LitStr(x), Ast.String
  | Ast.Binop(lexpr, op, rexpr) ->
    let lexprt = sast_expr_env lexpr in
    let rexprt = sast_expr_env rexpr in
    let (_, lt) = lexprt in
    let (_, rt) = rexprt in
    let opfailwith constraint_str =
      failwith (sprintf "Operator '%s' is only defined for %s (%s, %s found)"
        (Ast.string_of_op op) constraint_str (Ast.string_of_type lt) (Ast.string_of_type rt))
    in
    begin match op with
      | Ast.Add | Ast.Sub | Ast.Mul | Ast.Div ->
          if lt = rt && (lt = Ast.Float || lt = Ast.Int) then Sast.Binop(lexprt, op, rexprt), lt
          else opfailwith "float or int"

      | Ast.Mod ->
          if lt = rt && lt = Ast.Int then Sast.Binop(lexprt,op,rexprt), lt
          else opfailwith "int"

      | Ast.Eq | Ast.Neq ->
          if lt = rt then Sast.Binop(lexprt,op,rexprt), Ast.Bool
          else opfailwith "operands of the same type"

      | Ast.Lt | Ast.Lte ->
          if lt = rt && (lt = Ast.Float || lt = Ast.Int) then Sast.Binop(lexprt, op, rexprt), Ast.Bool
          else opfailwith "float or int"

    | Ast.And | Ast.Or ->
      if lt = rt && lt = Ast.Bool then Sast.Binop(lexprt,op,rexprt), Ast.Bool
      else opfailwith "bool"

    | Ast.Concat -> begin match lt, rt with
     (* also allow tracks to be concatted *)
     | Ast.Type("track"), Ast.Type("track") ->
       sast_expr_env (Ast.FunApply("ConcatTracks", [lexpr;rexpr]))
     | Ast.Array(l), Ast.Array(r) when l = r -> Sast.Binop(lexprt,op,rexprt), lt
     | Ast.String, Ast.String -> Sast.Binop(lexprt, op, rexprt), lt
     | _ -> opfailwith "strings, arrays, and tracks" end

    | Ast.Chord ->
      (* guarantee that chord binop is between two chords *)
      sast_expr_env (Ast.FunApply("ChordOfChords", [chord_of lexpr lt; chord_of rexpr rt]))

    | Ast.Octave ->
      let lexpr = match lt with
        | Ast.Type("pitch") | Ast.Array(Ast.Type("chord")) -> lexpr
        | Ast.Int -> Ast.FunApply("PitchOfInt", [lexpr])
        | _ -> failwith "octave only defined for pitch or int or chord list on left side"
      in begin match rt with
        | Ast.Int when lt = Ast.Array(Ast.Type("chord")) ->
            sast_expr_env (Ast.FunApply("OctaveChordList", [rexpr; lexpr]))
        | Ast.Int -> sast_expr_env (Ast.FunApply("AddPitchOctave", [lexpr;rexpr]))
        | _ -> failwith "octave only defined for int on right side"
      end

    | Ast.Zip ->
      let (fname, lhs, rhs) = begin match lt with
      |Ast.Float ->
        if(rt = Ast.Array(Ast.Type("chord"))) then
          ("ZipDiff", Ast.Arr([lexpr], Some(Ast.Float)), rexpr)
        else
          ("ZipSame", Ast.Arr([lexpr], Some(Ast.Float)),
                                Ast.Arr([(chord_of rexpr rt)], Some(Ast.Type("chord"))))
      |Ast.Array(Ast.Float) ->
```

```
            if rt = Ast.Array(Ast.Type("chord")) then
              ("ZipSame", lexpr, rexpr)
            else
              ("ZipDiff", lexpr, Ast.Arr([(chord_of rexpr rt)], Some(Ast.Type("chord"))))
        |_ -> failwith "left side expression of zip must of float or array of float"
      end
        in sast_expr_env (Ast.FunApply(fname, [lhs;rhs]))
  end
| Ast.Uniop(op, expr) ->
  let exprt = sast_expr_env expr in
  let (_, t) = exprt in
  begin match op with
    | Ast.Not when t = Ast.Bool -> Sast.Uniop(op, exprt), t
    | Ast.Not -> failwith "This operator is only defined for bool"
    | Ast.Neg when t = Ast.Int || t = Ast.Float -> Sast.Uniop(op, exprt), t
    | Ast.Neg -> failwith "This operator is only defined for int or float"
    | Ast.Sharp ->
        let expr = match t with
          | Ast.Int -> Ast.FunApply("PitchOfInt", [expr])
          | Ast.Type("pitch") -> expr
          | _ -> failwith "sharp is only defined for int or pitch"
        in sast_expr_env (Ast.FunApply("SharpPitch", [expr]))
    | Ast.Flat ->
        let expr = match t with
          | Ast.Int -> Ast.FunApply("PitchOfInt", [expr])
          | Ast.Type("pitch") -> expr
          | _ -> failwith "flat is only defined for int or pitch"
        in sast_expr_env (Ast.FunApply("FlatPitch", [expr]))
  end

| Ast.FunApply(name, arg_exprs) ->
  (* Common code for all function calls *)
  (* get typed versions of input expressions *)
  let arg_texprs = List.map arg_exprs ~f:(sast_expr_env) in
  (* get types of the typed arguments *)
  let arg_types = List.map arg_texprs ~f:(fun (_, t) -> t) in

```

```
    (* NH function calls *)
    if is_nh_function env name
    then
      (* find the function *)
      let num_args = List.length arg_exprs in
      let nh_fun_sig = find_function env.functions name num_args in
      let (_,Ast.FunDef(_,params,expr)) = nh_fun_sig in
      (* zip params with input types *)
      let tparams = match List.zip params arg_types with
        | None -> failwith "Internal error: Mismatched lengths of types and arguments while type checking function call"
        | Some(x) -> x
      in
      let has_seen_fun = (find_seen_function seen_funs name tparams) <> None in
      (* check if function type already inferred *)
      match List.find !tfuns_ref ~f:(fun (Sast.FunDef(n,tps,_)) -> name=n && tparams=tps) with
        (* already know function signature and not forced to re-infer subexpression types
          (or is forced but loop encountered, so need to use previous result anyway) *)
        | Some(Sast.FunDef(_,_,(_,t))) when (force && has_seen_fun) || not force ->
            Sast.FunApply(NhFunction(name), arg_texprs), t
        (* don't know function signature, or do know signature but
          forced to re-infer subexpression types *)
        | _ ->
          let seen_funs =
            if has_seen_fun
            (* if the function is already seen, we are in a loop - can't infer type;
              roll back to most recent conditional and see what we can do there
              (conditional catches Cant_infer_type exception) *)
            then raise
              (Cant_infer_type("Can't infer type of recursive call to nh function "^name))
            (* function not seen yet; we need to infer subexpression types, so mark this function as seen *)
            else (name,tparams):: seen_funs
          in
          let try_check_function_type force =
            (* forcing type inference means that cached results in
              tfuns_ref will be ignored unless a loop is encountered *)
            try check_function_type tparams expr tfuns_ref env seen_funs force
            with Failure(reason) ->
      Log.info "Function template type check failed. Inner exception: %s" reason;
      failwith ("Incorrect types passed into function "^name)
    in
    (* check if types of inputs can be used with this function *)
    let (sexpr, t) = try_check_function_type force in
    begin
    (* UPDATE tfuns_ref *)
    ignore(replace_add_fun tfuns_ref (Sast.FunDef(name, tparams, (sexpr, t))));
    (* check if it is safe to re-infer all subexpression types (ie tfun_ref is fully updated)
      and that we are not already trying to re-infer all subexpression types *)
    if List.length seen_funs = 1 && not force
     (* go back in to resolve all types;
      second pass guarantees no placeholder conditionals are in descendants *)
     then let (sexpr, t) = try_check_function_type true in
      ignore(replace_add_fun tfuns_ref (Sast.FunDef(name, tparams, (sexpr, t))));
      Sast.FunApply(NhFunction(name), arg_texprs), t
     (* pass sast back up the ast so higher expressions can infer type;
      could still have placeholder conditionals in descendants *)
     else Sast.FunApply(NhFunction(name), arg_texprs), t
    end

 (* C++ function calls *)
 else
  if is_cpp_function env name
  then
   (* find the function *)
   let decl = find_extern env.extern_functions name arg_types in
   let Ast.ExternFunDecl(cpp_file, cpp_ns, cpp_name, _, _, ret_type) = decl in
   (* If we got here, the function is OK *)
   Sast.FunApply(CppFunction(cpp_file, cpp_ns, cpp_name), arg_texprs), ret_type

  (* Function name doesn't exist *)
  else failwith ("There is no function named " ^ name)

| Ast.Block(exprs) ->
 let (texprs,_) = List.fold_left exprs ~init:([],env)
  ~f:(fun (texprs, env) expr ->
        (* propagate any env changes within block (due to new var initialization) *)
        let env =
          match texprs with
            | [] -> env
            | head::_ -> begin match head with
                | Sast.Init(name, (_, t), mutability), _ ->
                    let new_vars = (name, t, mutability) :: env.scope.variables in
                    let new_scope = { parent=env.scope.parent; variables=new_vars } in
                    { scope=new_scope; functions=env.functions;
                      extern_functions=env.extern_functions; types=env.types; }
                | _ -> env
              end
        in let texpr = sast_expr ~seen_funs:seen_funs ~force:force env tfuns_ref expr in
        (texpr :: texprs, env)
      )
    in
    let texprs = List.rev texprs in
    begin match List.last texprs with
      | Some(_, t) -> Block(texprs), t
      | None -> LitUnit, Ast.Unit
    end

  | Ast.VarRef(names) ->
      begin match names with
      | [] -> failwith "Internal error: VarRef(string list) had empty string list"
      | name :: fields ->
          try Sast.VarRef(names), find_ref_type env name fields
          with Not_found -> failwith (sprintf "%s referenced before initalization" (Ast.string_of_expr (VarRef(names))))
      end

  | Conditional(condition, case_true, case_false) ->
      let (condition, condition_t) = sast_expr_env condition in
      if condition_t <> Ast.Bool then
        failwith (sprintf "Condition must be a bool expression (%s found)" (Ast.string_of_type condition_t)) else

      let try_sast_expr_env expr =
        try Some(sast_expr_env expr)
```

```
          (* Note that Cant_infer_type is raised when an nh function has already
            been seen higher up in the ast and it’s not in tfuns_ref either *)
          (* Also note that this case won’t trigger if force was set to true (from FunApply) *)
          with Cant_infer_type(_) -> None
        in
        (* try to infer type of true branch *)
        begin match try_sast_expr_env case_true with
          (* true branch type inference failed,
            ie tfuns_ref is missing an nh function that has been used higher up in the ast *)
          | None ->
              (* see if other case can infer type *)
              begin match try_sast_expr_env case_false with
                (* neither branch terminates *)
                | None -> failwith "Couldn’t infer type of either branch of conditional"
                (* only false branch terminates, assume entire conditional is of that type;
                  return fake sast with correct type so that tfuns_ref can be updated *)
                | Some((_,t)) ->
                    let fake_sexpr = (Sast.LitUnit, t) in
                    Sast.Conditional((condition, condition_t), fake_sexpr, fake_sexpr), t
              end
          (* true branch type inference successful, now check false branch *)
          | Some((case_true, case_true_t)) ->
              (* see if other case can infer type *)
              begin match try_sast_expr_env case_false with
                (* both branch types have been inferred, check if types are the same *)
                | Some((case_false, case_false_t)) -> if case_true_t <> case_false_t
                    then failwith (sprintf "Both expressions in a conditional must have the same type (%s and %s found)"
                      (Ast.string_of_type case_true_t) (Ast.string_of_type case_false_t))
                    else Sast.Conditional( (condition, condition_t), (case_true, case_true_t), (case_false, case_false_t) ), case_true_
                (* false branch type inference failed, ie tfuns_ref is missing an nh function that
                    has been used higher up in the ast;
                  true branch terminates, assume entire conditional is of that type;
                  return fake sast with correct type so that tfuns_ref can be updated *)
                | None -> let fake_sexpr = (Sast.LitUnit, case_true_t) in
                    Sast.Conditional((condition, condition_t), fake_sexpr, fake_sexpr), case_true_t
              end
        end
```

```

| For(loop_var_name, items, body) ->
    let (items, items_t) = sast_expr_env items in
    let loop_var_t =
      match items_t with
      | Ast.Array(t) -> t
      | _ -> failwith (sprintf "You can only loop through an array (%s found)" (Ast.string_of_type items_t))
    in
    let env' = {
      scope = { variables = [ (loop_var_name, loop_var_t, Immutable) ]; parent = Some(env.scope) };
      functions = env.functions;
      extern_functions = env.extern_functions;
      types = env.types;
    } in
    let (body, body_t) = sast_expr env' tfuns_ref body in
    Sast.For((loop_var_name, loop_var_t), (items, items_t), (body, body_t)), body_t

| Throw(msg_expr) -> let (_,t) = sast_expr_env msg_expr in
    if t <> Ast.String then failwith "throw expects an expression of type string"
    else let msg = sast_expr_env (Ast.FunApply("PrintEndline",[msg_expr])) in
    Sast.Block([msg; (Sast.Exit(0), Ast.Unit)]), Ast.Unit

| Ast.Assign(names, expr, mutability) ->
    (* Type-check RHS of the assignment *)
    let (value, tvalue) = sast_expr_env expr in
    begin
      match names with
      (* No variable name *)
      | [] -> failwith "Internal error: Assign(names, _, _) had empty string list"
      | name :: fields ->
          try
            (* Check that variable is mutable *)
            let (_, _, mutability) = find_variable env.scope name in
            if mutability = Immutable then failwith (sprintf "cannot assign to immutable %s" name) else
            (* Check that types match *)
            let t = find_ref_type env name fields in
            if t <> tvalue then failwith (sprintf "cannot assign type %s to %s (type %s)"
       (Ast.string_of_type tvalue) (Ast.string_of_expr (VarRef(names))) (Ast.string_of_type t)) else
      (* Passed all checks! *)
      Sast.Assign(names, (value, tvalue)), Ast.Unit
     with Not_found ->
      match fields with
      | [] -> Init(name, (value, tvalue), mutability), Ast.Unit
      | _ -> failwith ("Cannot assign to fields of uninitialized var " ^ (Ast.string_of_expr (VarRef(names))))
  end

| Ast.StructInit(typename, init_list) ->
  let TDefault(_, defaults) = match List.find env.types ~f:(fun (TDefault(n,_)) -> typename = n) with
   | Some(x) -> x
   | None -> failwith ("type "^typename^" not found")
  in
  let fields = List.map defaults ~f:(fun (n,(_,t)) -> (n,t)) in
  let sexprs = List.map init_list ~f:(sast_expr_env) in
  let varname = function
   | Sast.Init(name, (_, t), _), _
     when begin match List.find fields ~f:(fun (n,_) -> n = name) with
      | Some((_,tfield)) when tfield = t -> true
      | _ -> false
     end
     -> name
   | Assign(name::tail,(_,t)),_
     when List.length tail = 0 && begin match List.find fields ~f:(fun (n,_) -> n = name) with
      | Some((_,tfield)) when tfield = t -> true
      | _ -> false
     end
     -> name
   | _ -> failwith ("Only assignments of fields are allowed in type init of "^typename)
  in
  if List.contains_dup sexprs
   ~compare:(fun lsexpr rsexpr ->
    let ln = varname lsexpr and rn = varname rsexpr in
    compare ln rn
   )
   then failwith ("cannot assign fields multiple times in type init of "^typename)
```

```
    else let init_exprs = List.fold_left defaults ~init:[]
     ~f:(fun init_exprs (name, expr) ->
      (* grab default if not explicitly initalized *)
      let field_expr = function
       | Init(_, expr, _), _ -> expr
       | Assign(_, expr), _ -> expr
       | _ -> failwith ("Internal error: non init/assign sexpr found in type init")
      in
      match List.find sexprs ~f:(fun sexpr -> name = varname sexpr) with
       | Some(x) -> (name, field_expr x) :: init_exprs
       | None -> (name, expr) :: init_exprs
     )
    in
    Struct(typename, init_exprs), Ast.Type(typename)

  | Ast.Arr(exprs, Some(t)) ->
    let texprs = List.map exprs ~f:sast_expr_env in
    let types_same = List.for_all texprs ~f:(fun (_, item_t) -> t = item_t) in
    if not types_same then failwith (sprintf "Array has inconsistent types (expected %s)" (Ast.string_of_type t)) else
    Sast.Arr(texprs, t), Ast.Array(t)

  | Ast.Arr(exprs, None) ->
    let infer_t = (
     match exprs with
     | [] -> failwith "Internal error: parser gave untyped empty array"
     | head :: _ -> let (_, t) = sast_expr_env head in t
    ) in
    sast_expr_env (Ast.Arr(exprs, Some(infer_t)))

  | Ast.ArrMusic(exprs) ->
    (match exprs with
     | [] -> failwith "Internal error: parser gave untyped empty array"
     | head :: _ ->
       (* Infer the type of the array *)
       let (_, infer_t) = sast_expr_env head in
       match infer_t with
       (* Durations (aka float) are handled normally *)
```

```
            | Ast.Float -> sast_expr_env (Ast.Arr(exprs, Some(infer_t)))

            (* For int, pitch, or chord, promote everything to chord *)
            | Ast.Int | Ast.Type("pitch") | Ast.Type("chord") ->
                (* First pass: get original types *)
                let texprs = List.map exprs ~f:(fun expr -> let (_, t) = sast_expr_env expr in (expr, t)) in
                (* Second pass: promote everything to chord *)
                let texprs = List.map texprs ~f:(fun (expr, item_t) ->
                  let expr = try chord_of expr item_t with Failure(_) ->
                      failwith (sprintf "Music array has inconsistent item of type %s (expected %s)"
                        (Ast.string_of_type item_t) (Ast.string_of_type infer_t))
                  in
                  sast_expr_env expr
                ) in
                let t = Ast.Type("chord") in
                Sast.Arr(texprs, t), Ast.Array(t)

            | _ -> failwith (sprintf "Cannot use music array literal with type %s" (Ast.string_of_type infer_t))

      )

  | Ast.ArrIdx(id_var, expr) ->
    let (exp, t) = sast_expr env tfuns_ref expr in
      if t <> Ast.Int then
        failwith(sprintf "Array Index must be an integer (%s found)" (Ast.string_of_type t))
      else
        let (_, t_v) = sast_expr env tfuns_ref (Ast.VarRef(id_var)) in
        match t_v with
        |Ast.Array(x) -> Sast.ArrIdx(id_var, (exp, t)), x
        |_ -> failwith("Cannot index into a non-array object")

and check_function_type tparams expr tfuns_ref env seen_funs force =
  let tparams' = List.map tparams ~f:(fun (name, t) -> (name, t, Ast.Mutable)) in
  let env' = {
    (* allow global scope *)
    scope = { variables = tparams'; parent = Some(get_top_scope env.scope) };
    functions = env.functions;
    extern_functions = env.extern_functions;
    types = env.types;
  } in
  sast_expr ~seen_funs:seen_funs ~force:force env' tfuns_ref expr

and typed_typedefs env tfuns_ref typedefs =
  (* Assuming the users have defined the types in the correct order *)
  let (tdefaults, _) = List.fold_left typedefs ~init:([],env)
    ~f:(fun (tdefaults, env) (Ast.TypeDef(name, exprs)) ->
      let sexprs = List.map exprs ~f:(sast_expr env tfuns_ref) in
      let fields = List.map sexprs
        ~f:(fun sexpr ->
          match sexpr with
            | Init(name, expr, _), _ -> (name, expr)
            | Assign(name::tail, expr),_ when List.length tail = 0 -> (name, expr)
            | _ -> failwith ("Only initialization of fields are allowed in type decl of "^name)
        )
      in
      if List.contains_dup fields
        ~compare:(fun (ln,_) (rn,_) -> compare ln rn)
        then failwith ("Cannot init fields multiple times in type decl of "^name)
      else
      let tdefaults = TDefault(name, fields)::tdefaults in
      let env =
        {
          scope = env.scope;
          functions = env.functions;
          extern_functions = env.extern_functions;
          types = tdefaults;
        }
      in tdefaults, env
    )
  in
  (* Remove repeats ... hope the user knows what he was doing... *)
  let tdefaults = List.dedup tdefaults
    ~compare:(fun (TDefault(ln,_)) (TDefault(rn,_)) -> compare ln rn)
  in
  (* Build dependency graph *)
  let type_deps = List.map tdefaults
    ~f:(fun (TDefault(name, defaults)) ->
      let fields = List.map defaults ~f:(fun (_,(_,t)) -> t) in (Ast.Type(name), fields)
    )
  in
  if has_cycle type_deps then failwith ("No mutually recursive types allowed")
  else tdefaults

and typed_externs externfuns env_types =
  List.fold_left externfuns ~init:[]
    ~f:(fun validated item ->
      let Ast.ExternFunDecl(_, _, _, _, arg_types, ret_type) = item in
      (* Find out whether the user-defined types, if any, are valid *)
      if List.for_all (ret_type :: arg_types) ~f:(function
        | Ast.Array(Ast.Type(name)) | Ast.Type(name) ->
          List.exists env_types ~f:(fun (TDefault(type_name, _)) -> type_name = name)
        | _ -> true (* built-in types are always allowed *)
      )
      then
        (* Only add new things *)
        if List.mem validated item
        then failwith ("External function has already been declared:\n" ^ Ast.string_of_extern item)
        else item :: validated
      else failwith ("Invalid type in external function declaration:\n" ^ Ast.string_of_extern item)
    )

let rec verify_no_fun_ast ast =
  let verify_all exprs =
    List.iter exprs ~f:verify_no_fun_ast
  in match ast with
    | Ast.FunApply(_,_) -> failwith "Function found"
    | LitBool(_) | LitFloat(_) | LitInt(_) | LitStr(_) | VarRef(_) -> ()
    | Binop(lexpr,_,rexpr) | For(_,lexpr,rexpr) -> verify_all [lexpr; rexpr]
    | Uniop(_,expr) | ArrIdx(_,expr) | Throw(expr) | Assign(_,expr,_) -> verify_all [expr]
    | Conditional(bexpr,texpr,fexpr) -> verify_all [bexpr; texpr; fexpr]
    | Arr(exprs,_) | ArrMusic(exprs) | Block(exprs) | StructInit(_,exprs) -> verify_all exprs
```

```ocaml

(* Note that includes have been processed and merged into exprs by this point *)
let sast_of_ast (fundefs, externs, exprs, typedefs) =
  (* ensure there are no function calls in typedefs *)
  List.iter typedefs ~f:(fun (Ast.TypeDef(name,exprs)) ->
    try List.iter exprs ~f:verify_no_fun_ast
    with Failure(_) -> failwith ("Function call found in typedef "^name)
  );
  (* make sure fundefs are unique *)
  let nfundefs = check_unique_functions fundefs externs in
  (* temporary env for evaluating tdefaults *)
  let temp_env = {
    scope = { variables=[]; parent=None };
    functions = nfundefs;
    extern_functions = externs;
    types = [];
  } in
  let tfuns_ref = ref [] in
  let tdefaults = typed_typedefs temp_env tfuns_ref typedefs in
  let externs = typed_externs externs tdefaults in
  let env = {
    scope = { variables=[]; parent=None };
    functions = nfundefs;
    extern_functions = externs;
    types = tdefaults;
  } in
  let sexpr = sast_expr env tfuns_ref (Ast.Block(exprs)) in
  let cpp_includes = List.dedup (List.map externs ~f:(fun (ExternFunDecl(header, _, _, _, _, _)) -> header)) in
  cpp_includes, !tfuns_ref, sexpr, env.types
```

9.2.43 src/update_version.sh

```bash
#!/bin/bash

GIT_HASH=‘git describe --abbrev --dirty --always‘
GIT_BRANCH=‘git rev-parse --abbrev-ref HEAD‘
GIT_TAG=‘git describe --abbrev=0 --tags‘

```

```
OUT_PATH='version.ml'

echo "(* Build script automatically updates these strings *)" > $OUT_PATH
echo "let release () = \"$GIT_TAG\"" >> $OUT_PATH
echo "let build () = \"$GIT_HASH on branch $GIT_BRANCH\"" >> $OUT_PATH
```

9.2.44 src/version.mli

```
val release : unit -> string
val build : unit -> string
```

9.2.45 style/note-hashtag.tmLanguage

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>fileTypes</key>
        <array>
                <string>nh</string>
        </array>

        <key>name</key>
        <string>Note Hashtag</string>

        <key>patterns</key>
        <array>
                <dict>
                        <key>comment</key>
                        <string>Line Comments</string>
                        <key>match</key>
                        <string>//.*</string>
                        <key>name</key>
                        <string>comment.line.double-slash.nh</string>
                </dict>
                <dict>
                        <key>comment</key>
                        <string>Block Comments</string>
                <key>include</key>
                <string>#multi_line_comment</string>
        </dict>
        <dict>
                <key>comment</key>
                <string>Strings</string>
                <key>match</key>
                <string>\"((\\\")|[^\"])*\"</string>
                <key>name</key>
                <string>string.quoted.double</string>
        </dict>
        <dict>
                <key>comment</key>
                <string>Language literals</string>
                <key>match</key>
                <string>(\b(true|false|~)\b)|(\(\s*\))</string>
                <key>name</key>
                <string>constant.language.nh</string>
        </dict>
        <dict>
                <key>match</key>
                <string>\b(e|eighth|q|quarter|h|half|w|whole|t|triplet)\b</string>
                <key>name</key>
                <string>support.constant.nh</string>
        </dict>
        <dict>
                <key>match</key>
                <string>\b(key_sig|time_sig|tempo)\b</string>
                <key>name</key>
                <string>variable.language.nh</string>
        </dict>
        <dict>
                <key>match</key>
                <string>([-+]?\d*\.?\d+)(e(\+?|-)\d+)?</string>
                <key>name</key>
                <string>constant.numeric.nh</string>
        </dict>
        <dict>
            <key>comment</key>
            <string>Built in types</string>
            <key>match</key>
            <string>\b(int|string|float|unit|bool|fun|type)\b</string>
            <key>name</key>
            <string>storage.type.nh</string>
        </dict>
        <dict>
            <key>match</key>
            <string>\b(key_signature|time_signature|pitch|chord|track|song)\b</string>
            <key>name</key>
            <string>storage.type.nh</string>
        </dict>
        <dict>
            <key>comment</key>
            <string>Modifiers</string>
            <key>match</key>
            <string>(const)</string>
            <key>name</key>
            <string>storage.modifier.nh</string>
        </dict>
        <dict>
            <key>comment</key>
            <string>Operator keywords</string>
            <key>match</key>
            <string>\b(init|beget|bringintobeing|-&gt;)\b</string>
            <key>name</key>
            <string>keyword.operator.other.nh</string>
        </dict>
        <dict>
            <key>comment</key>
            <string>Conditional Control Keywords</string>
            <key>match</key>
            <string>\b(if|then|else|be|unless|inwhichcase)\b</string>
            <key>name</key>
            <string>keyword.control.conditional.nh</string>
```

```
                </dict>
                <dict>
                        <key>comment</key>
                        <string>Loop Control Keywords</string>
                        <key>match</key>
                        <string>\b(for|in|do)\b</string>
                        <key>name</key>
                        <string>keyword.control.loop.nh</string>
                </dict>
                <dict>
                        <key>comment</key>
                        <string>Other control keywords</string>
                        <key>match</key>
                        <string>\b(include|extern|throw)\b</string>
                        <key>name</key>
                        <string>keyword.control.other.nh</string>
                </dict>
                <dict>
                        <key>match</key>
                        <string>(\|\||\&amp;\&amp;|==|!=|!|&lt;=|&gt;=|&gt;|&lt;)</string>
                        <key>name</key>
                        <string>keyword.operator.bool.nh</string>
                </dict>
                <dict>
                        <key>match</key>
                        <string>(|\+|\-|\*|/|%)</string>
                        <key>name</key>
                        <string>keyword.operator.arithmetic.nh</string>
                </dict>
                <dict>
                        <key>comment</key>
                        <string>Variables</string>
                        <key>match</key>
                        <string>\b[a-z_][A-Za-z0-9_]*\b</string>
                        <key>name</key>
                        <string>variable.other.nh</string>
                </dict>
                <dict>
                        <key>comment</key>
                        <string>Functions</string>
                        <key>match</key>
                        <string>\b[A-Z][A-Za-z0-9_]*\b</string>
                        <key>name</key>
                        <string>entity.name.function.nh</string>
                </dict>
                <dict>
                        <key>match</key>
                        <string>(#|:|@|b|,)</string>
                        <key>name</key>
                        <string>keyword.operator.musical.nh</string>
                </dict>
                <dict>
                        <key>match</key>
                        <string>\b(\.)\b|(=|,|\$)</string>
                        <key>name</key>
                        <string>keyword.operator.other.nh</string>
                </dict>
        </array>

    <key>repository</key>
    <dict>
        <!-- Based on https://github.com/tijn/sublime-waxeye/commit/54cfd17ee17569f8cdbd6f0c15adea4b2a19525b -->
        <key>multi_line_comment</key>
        <dict>
            <key>begin</key>
            <string>\/\*</string>
            <key>end</key>
            <string>\*\/</string>
            <key>name</key>
            <string>comment.block.nh</string>
            <key>patterns</key>
            <array>
                <dict>
                    <key>include</key>
                    <!-- recurse to support nested comments -->
                    <string>#multi_line_comment</string>
                </dict>
            </array>
        </dict>
    </dict>

        <key>scopeName</key>
        <string>source.nh</string>

        <key>uuid</key>
        <string>7f6b19e8-b940-426b-b77b-a2593befdc39</string>
</dict>
</plist>
```

9.2.46 style/styletest.nh

```

include mylib
extern something -> int
throw error
hello why are we here
Fhi
type hi = {
  x = 5
}
init hi
bringintobeing hi
beget hi
for x in z do
if then else
be unless inwhichcase
myvar4b
1b 2b 3# 4# 5b
hi:hello
3@2
hello.hi
const fillo
```

```
int{}
int
string
float
unit
bool
// asdf
fun Hello arg arg arg = (
  code in here
)
hi || hello && why + hi - * / % == != ! = > < >= <=
bye
bollocksby
hib
"hello \n my name"
"is \" mike"
"really" magic "mike"
1
1.1
.2
1e4
2e-3
3.3e+5
0.43e23
.43e1
5,6 3,4@4,2 2,1 $ $ asdf$efe
{}
[2 3 4]
() (   )
pitch[]
chord[]
track{}
song{}
true
false
_myName
q quarter h half w whole t triplet
```

```
key_signature
```

9.2.47 test/array_basic.nh

```
x = { 1 2 3 }
PrintEndline (StringOfInt x.(0))
PrintEndline (StringOfInt x.(1))
PrintEndline (StringOfInt x.(2))

y = { 4.0 4.5 5.0 }
PrintEndline (StringOfFloat y.(0))
PrintEndline (StringOfFloat y.(1))
PrintEndline (StringOfFloat y.(2))

z = { "6 str" "7 str" "8 str" }
PrintEndline z.(0)
PrintEndline z.(1)
PrintEndline z.(2)

u = { () () () }
if u.(0) == () then PrintEndline "array of unit OK" else PrintEndline ":("

type ty = {
  a = "hi"
}
ta = { (init ty) (init ty) }
ti = ta.(0)
PrintEndline ti$a
```

9.2.48 test/array_basic.out

```
1
2
3
4.000000
4.500000
5.000000
6 str
```

```
7 str
8 str
array of unit OK
hi
```

9.2.49 test/array_empty.nh

```
a_1 = int{}
a_2 = float{}
a_3 = string{}
a_4 = unit{}
a_5 = bool{}

type ty = {
  a = "hi"
}
a_6 = ty{}
```

9.2.50 test/array_empty.out

9.2.51 test/array_equality.nh

```
x = [1 2 3]
y = [1 2 3]
if x == y then
PrintEndline "yes"
else
PrintEndline "no"
```

9.2.52 test/array_equality.out

```
yes
```

9.2.53 test/array_music_chords.nh

```

things = [ 1 1,3 2 ]

```

```
c = things.(0)
p = c$pitches.(0);
Print (StringOfInt (p$rank))

c = things.(1)
p = c$pitches.(0);
Print (StringOfInt (p$rank))
p = c$pitches.(1);
Print (StringOfInt (p$rank))

c = things.(2)
p = c$pitches.(0);
Print (StringOfInt (p$rank))
```

9.2.54 test/array_music_chords.out

```
1132
```

9.2.55 test/array_music_float.nh

```
x = [1.0 2.0 3.0 4.0]
PrintEndline(StringOfFloat(x.(0)))
PrintEndline(StringOfFloat(x.(1)))
PrintEndline(StringOfFloat(x.(2)))
PrintEndline(StringOfFloat(x.(3)))
```

9.2.56 test/array_music_float.out

```
1.000000
2.000000
3.000000
4.000000
```

9.2.57 test/array_music_octaves.nh

```

things = [ 1 1@1 1 ]

```

```
for c in things do (
  p = c$pitches.(0);
  Print (StringOfInt (p$rank))
  Print (StringOfInt (p$octave))
)
```

9.2.58 test/array_music_octaves.out

```
101110
```

9.2.59 test/array_music_pitch.nh

```
x = [ 1 2 3 4 ]

y = x.(0)
z = y$pitches.(0)
Print (StringOfInt z$rank)

y = x.(1)
z = y$pitches.(0)
Print (StringOfInt z$rank)

y = x.(2)
z = y$pitches.(0)
Print (StringOfInt z$rank)

y = x.(3)
z = y$pitches.(0)
Print (StringOfInt z$rank)
```

9.2.60 test/array_music_pitch.out

```
1234
```

9.2.61 test/array_music_preoctave.nh

```

things = @3[1 2 3]
```

```

for c in things do (
  p = c$pitches.(0);
  Print (StringOfInt (p$rank))
  Print (StringOfInt (p$octave))
)
```

9.2.62 test/array_music_preoctave.out

```
132333
```

9.2.63 test/array_nested.nh

```
x = {{1}}
y = x.(0)
PrintEndline(StringOfInt(y.(0)))
```

9.2.64 test/array_nested.out

```
1
```

9.2.65 test/assign_const.nh

```
x = 0
x = 42
PrintEndline (StringOfInt x)
const y = 10
PrintEndline (StringOfInt y)
```

9.2.66 test/assign_const.out

```
42
10
```

9.2.67 test/assign_retval.nh

```
y = "why"
x = (y = "what")
Print y
```

```

x
```

9.2.68 test/assign_retval.out

```
what
```

9.2.69 test/comment_multiline.nh

```
Print "hi\n"
/* Print "bye\n" */

/*
/*
Print "poo\n"
// I can write anything here
*/*/
Print "hello\n"
```

9.2.70 test/comment_multiline.out

```
hi
hello
```

9.2.71 test/comment_multiline_no_nl.nh

```
Print "hi\n"
/*
 * WARNING: Be careful when editing this file in vim, etc. You must remove the newline after the last line (invisible in
 * some editors).
 */
/* Print "test no newlines\n" */
```

9.2.72 test/comment_multiline_no_nl.out

```
hi
```

9.2.73 test/comment_oneline.nh

```
Print "should print\n"
// PrintEndline "shouldn't print"
//comment with no space after slashes
Print "this too\n"
```

9.2.74 test/comment_oneline.out

```
should print
this too
```

9.2.75 test/comment_oneline_no_nl.nh

```
Print "hi\n"
// WARNING: Be careful when editing this file in vim, etc. You must remove the newline after the last line (invisible in
// some editors).
// comment with no newline after it
```

9.2.76 test/comment_oneline_no_nl.out

```
hi
```

9.2.77 test/demo_twinkle.nh

```

intro = 0.25:[ 1 1 5 5 6 6 ] . 0.5:5
chorus = Rhythms intro : [ 4 4 3 3 2 2 1 ]
bridge = Relative 1 chorus

// the tune
twinkle = intro . chorus . bridge . bridge . intro . chorus

full_twinkle = [1 1 5 5 6 6 5 4 4 3 3 2 2 1 5 5 4 4 3 3 2 5 5 4 4 3 3 2\
                  1 1 5 5 6 6 5 4 4 3 3 2 2 1]

if twinkle$chords == full_twinkle then Print "TWINKLE" else Print "TINKLE"
```

9.2.78 test/demo_twinkle.out

```
TWINKLE
```

9.2.79 test/flatsharp.nh

```


(
x = [ 1b 2# ]
p1 = init pitch {rank=1; offset=-1}
p2 = init pitch {rank=2; offset=1}
y = p2#b b
p3 = init pitch {rank=2; offset = 0}
if x == [p1 p2] && y == p3 then
  Print "Flat and Sharp work" else Print "Flat and Sharp don't work"
)
```

9.2.80 test/flatsharp.out

```
Flat and Sharp work
```

9.2.81 test/for_multiline.nh

```
for n in { "3" "2" "1" } do (
  Print "poo "
  PrintEndline n
)
for i in { 1 (1 + 1) (1 + 1 + 1) } do (
  Print "poo "
  PrintEndline (StringOfInt i)
)
```

9.2.82 test/for_multiline.out

```
poo 3
poo 2
poo 1
poo 1
poo 2
poo 3
```

9.2.83 test/for_nested.nh

```
for n in { "a: " "b: " "c: " } do (
  Print n
  for n in { 1 2 3 } do (
    Print (StringOfInt n)
  )
  Print "\n"
)

for n in { 10 100 } do for m in { 1 2 3 } do PrintEndline (StringOfInt (n * m))
```

9.2.84 test/for_nested.out

```
a: 123
b: 123
c: 123
10
20
30
100
200
300
```

9.2.85 test/for_simple.nh

```
for n in { "3" "2" "1" } do PrintEndline n
for i in { 1 (1 + 1) (1 + 1 + 1) } do PrintEndline (StringOfInt i)
```

9.2.86 test/for_simple.out

```
3
2
1
1
2
3
```

9.2.87 test/fundef_mutual_rec.nh

```

fun First x = (
  Print (StringOfInt x)
  if x > 1 then (Print "s"; Second x; "not") else "done"
)

fun Second x = (
  Print (First (x-1))
)

Print (First 5)
```

9.2.88 test/fundef_mutual_rec.out

```
5s4s3s2s1donenotnotnotnot
```

9.2.89 test/fundef_overloaded.nh

```
fun MyFavorite x y = if x > y then x else y

PrintEndline (StringOfInt (MyFavorite 1 2))
PrintEndline (StringOfFloat (MyFavorite 1.0 2.0))
```

9.2.90 test/fundef_overloaded.out

```
2
2.000000
```

9.2.91 test/fundef_rec.nh

```

fun Factorial x = if x <= 1 then 1 else x * Factorial (x-1)

x = Factorial 3
Print (StringOfInt x)
```

9.2.92 test/fundef_rec.out

```
6
```

9.2.93 test/fundef_rec_notail.nh

```

fun Factorial x = if x > 1 then x * Factorial (x-1) else 1

x = Factorial 3
Print (StringOfInt x)
```

9.2.94 test/fundef_rec_notail.out

```
6
```

9.2.95 test/fundef_simple.nh

```

fun Hello x = x

PrintEndline (Hello "what")
PrintEndline (StringOfInt (Hello 1))
```

9.2.96 test/fundef_simple.out

```
what
1
```

9.2.97 test/fundef_type.nh

```
type ty = {
  mem = "hi"
}
fun Ident x = x
x = Ident (init ty)
Print x$mem
```

9.2.98 test/fundef_type.out

```
hi
```

9.2.99 test/globals_fundef_simple.nh

```

top = "top"

fun Hello x = (
  Print x
  Print top
)

Hello "u"
```

9.2.100 test/globals_fundef_simple.out

```
utop
```

9.2.101 test/if_as-expr.nh

```
PrintEndline (if true then "camels" else "even-toed ungulates")
PrintEndline (
  if true then (
    "hunter2"
  ) else (
    "*******"
  )
)
```

9.2.102 test/if_as-expr.out

```
camels
hunter2
```

9.2.103 test/if_innovations.nh

```
be PrintEndline "This revolutionary new language construct" unless false inwhichcase PrintEndline "Fail 1"
be (
  PrintEndline "is great for keeping programmers on their toes"
) unless false inwhichcase (
  PrintEndline "Fail 2"
)
```

9.2.104 test/if_innovations.out

```
This revolutionary new language construct
is great for keeping programmers on their toes
```

9.2.105 test/if_multiline.nh

```
if true then (
  Print "Pass 1"
  Print "\n"
) else (
  PrintEndline "Fail 1"
)

if false
then
(
  PrintEndline "Fail 2"
)
else
(
  Print "Pass 2"
  Print "\n"
)

if

true

then

(

  Print "Pass 3"

  Print "\n"

)

else

(

  PrintEndline "Fail 3"

)
```

9.2.106 test/if_multiline.out

```
Pass 1
Pass 2
Pass 3
```

9.2.107 test/if_nested.nh

```
if true then (
  if true then
    PrintEndline "Pass 1"
  else
    PrintEndline "Fail 1 (inner else)"
)
else PrintEndline "Fail 1 (outer else)"

if false then PrintEndline "Fail 2 (outer if)"
else (
  if false then
    PrintEndline "Fail 2 (inner if)"
  else
    PrintEndline "Pass 2"
)
```

9.2.108 test/if_nested.out

```
Pass 1
Pass 2
```

9.2.109 test/if_simple.nh

```
if true then PrintEndline "it works" else PrintEndline "you suck"
PrintEndline "this should print too"
```

9.2.110 test/if_simple.out

```
it works
this should print too
```

9.2.111 test/lit_bool.nh

```
if true then PrintEndline "Hello" else PrintEndline "World"
if false then PrintEndline "Hello" else PrintEndline "World"
```

9.2.112 test/lit_bool.out

```
Hello
World
```

9.2.113 test/nested_types.nh

```

type inside = {
  a = "in here!"
}

type outside = {
  a = init inside
}

inception = init outside

Print inception$a$a
```

9.2.114 test/nested_types.out

```
in here!
```

9.2.115 test/ops_arith.nh

```
PrintEndline (StringOfInt (1 + 1))
PrintEndline (StringOfInt (4 - 2))
PrintEndline (StringOfInt (6 * 7))
PrintEndline (StringOfInt (5 / 2))
PrintEndline (StringOfInt (5 % 2))

PrintEndline (StringOfFloat (1.0 + 1.0))
PrintEndline (StringOfFloat (4.0 - 2.0))
PrintEndline (StringOfFloat (6.0 * 7.0))
PrintEndline (StringOfFloat (5.0 / 2.0))
```

9.2.116 test/ops_arith.out

```
2
2
42
2
1
2.000000
2.000000
42.000000
2.500000
```

9.2.117 test/ops_bool.nh

```
if true || false then PrintEndline "Or :)" else PrintEndline "Or :("
be PrintEndline "And :)" unless true && false inwhichcase PrintEndline "And :("
if !false then PrintEndline "Not :)" else PrintEndline "Not :("
if !!true then PrintEndline "Double Not :)" else PrintEndline "Double Not :("
if !false && !!true then PrintEndline "Not in expressions :)" else PrintEndline "Not in expressions :("
if false && false || true then PrintEndline "Precedence :)" else PrintEndline "Precedence :("
if false && (false || true) then PrintEndline "Precedence override :(" else PrintEndline "Precedence override :)"
```

9.2.118 test/ops_bool.out

```
Or :)
And :)
Not :)
Double Not :)
Not in expressions :)
Precedence :)
Precedence override :)
```

9.2.119 test/ops_compare.nh

```
if -42.0 < 42.0 then PrintEndline "-42.0 < 42.0" else PrintEndline "failed float lt"
if 42.0 > -42.0 then PrintEndline "42.0 > -42.0" else PrintEndline "failed float gt"
if -42.0 <= 42.0 then PrintEndline "-42.0 <= 42.0" else PrintEndline "failed float lte"
if 42.0 >= -42.0 then PrintEndline "42.0 >= -42.0" else PrintEndline "failed float gte"

if 1000 > 0 then PrintEndline "1000 > 0" else PrintEndline "failed int gt"
if -1000 < 0 then PrintEndline "-1000 < 0" else PrintEndline "failed int lt"
if 1 >= 1 then PrintEndline "1 >= 1" else PrintEndline "failed int gte 1"
if 2 >= 1 then PrintEndline "2 >= 1" else PrintEndline "failed int gte 2"
if 1 <= 1 then PrintEndline "1 <= 1" else PrintEndline "failed int lte 1"
if 1 <= 2 then PrintEndline "1 <= 2" else PrintEndline "failed int lte 2"
```

9.2.120 test/ops_compare.out

```
-42.0 < 42.0
42.0 > -42.0
-42.0 <= 42.0
42.0 >= -42.0
1000 > 0
-1000 < 0
1 >= 1
2 >= 1
1 <= 1
1 <= 2
```

9.2.121 test/ops_concat_array.nh

```
PrintEndline (if { 1 2 } . { 3 4 } == { 1 2 3 4 } then "concat arrays ok" else "concat arrays no")
```

```

arr = { 1 2 }
PrintEndline (if arr . arr == { 1 2 1 2 } then "concat arrays to self ok" else "concat ararys to self :(")

PrintEndline (if arr . int{} == arr then "concat empty ok" else "concat empty :(")
```

9.2.122 test/ops_concat_array.out

```
concat arrays ok
concat arrays to self ok
concat empty ok
```

9.2.123 test/ops_concat_string.nh

```
PrintEndline ("Hello " . "World")
```

9.2.124 test/ops_concat_string.out

```
Hello World
```

9.2.125 test/ops_equality.nh

```
if 42 == 42 then PrintEndline "42 == 42" else PrintEndline "42 != 42"
be PrintEndline "-42 != 42" unless -42 == 42 inwhichcase PrintEndline "-42 == 42"
if true == true then PrintEndline "world will not end today" else PrintEndline "world may end soon"
if false != true then PrintEndline "all ok" else PrintEndline "felt a disturbance in the force"
if "foo" == "foo" then PrintEndline "strings gonna string" else PrintEndline "strings :("
if "foo" != "Foo" then PrintEndline "string neq OK" else PrintEndline "strings :(("
```

9.2.126 test/ops_equality.out

```
42 == 42
-42 != 42
world will not end today
all ok
strings gonna string
string neq OK
```

9.2.127 test/print_hello.nh

```
Print "Hello "
PrintEndline "World"
```

9.2.128 test/print_hello.out

```
Hello World
```

9.2.129 test/print_nums.nh

```
PrintEndline (StringOfFloat 42.0)
PrintEndline (StringOfInt 42)
PrintEndline (StringOfFloat (Pow 4.0 2.0))
PrintEndline (StringOfBool true)

PrintInt (-42); Print "\n"
PrintFloat (-42.0); Print "\n"
PrintBool false; Print "\n"
```

9.2.130 test/print_nums.out

```
42.000000
42
16.000000
true
-42
-42.000000
false
```

9.2.131 test/simple_assign.nh

```

x = "x"
y = 10

Print x
Print "\n"
Print (StringOfInt y)
```

9.2.132 test/simple_assign.out

```
x
10
```

9.2.133 test/simple_block.nh

```

(
  asdf = 5
)

g = 5
(
  g = 6
)
```

9.2.134 test/simple_block.out

9.2.135 test/simple_varref.nh

```

x = 5
x
```

9.2.136 test/simple_varref.out

9.2.137 test/std_arpeggio.nh

```
p1 = PitchOfInt 1
p2 = PitchOfInt 2
p3 = PitchOfInt 3
p4 = PitchOfInt 4
p_list = {p1 p2 p3 p4}
c = 1,2,3,4
a = (Arpeggio c)
if a == p_list then
  PrintEndline "Arpeggio works"
else
  PrintEndline "Arpeggio doesn't work"
```

9.2.138 test/std_arpeggio.out

```
Arpeggio works
```

9.2.139 test/std_chords.nh

```
p = [1 5 6 7 1]
r = [1.0 2.0 1.0 0.5 2.2]
tr = init track{chords = p; durations = r}
out_p = Chords tr
if out_p == p then
  PrintEndline "Chords works"
else
  PrintEndline "Chords doesn't work"
```

9.2.140 test/std_chords.out

```
Chords works
```

9.2.141 test/std_par_tracks.nh

```

(

p1 = [1 2]
r1 = [1.0 1.0]
t1 = init track{chords = p1; durations = r1}
p2 = [3 5]
r2 = [1.0 1.0]
t2 = init track{chords = p2; durations = r2}

s = Parallel { t1 t2 }
if s$tracks.(0) == {t1} then Print "G" else Print "B"

if s$tracks.(1) == {t2} then Print "G" else Print "B"

)
```

9.2.142 test/std_par_tracks.out

```
GG
```

9.2.143 test/std_range_fun.nh

```
x = Range 1 3
for i in x do PrintEndline (StringOfInt i)

if Range 1 1 == int{} then PrintEndline "empty range" else PrintEndline "you sucks"
```

9.2.144 test/std_range_fun.out

```
1
2
empty range
```

9.2.145 test/std_render.nh

```
tr = init track { durations = [ q q q q q q w ]; chords = [ 1 2 3 4 5 (init chord) (1,3,5) ] }
so = init song { tracks = { { tr } }; volumes = { 1.0 } }
Render so "test-render.wav"
PrintEndline "I didn't crash and now you have a song!"
```

9.2.146 test/std_render.out

```
I didn't crash and now you have a song!
```

9.2.147 test/std_render_flatsharp.nh

```

asdf = quarter:[ 1# 2b ]

Render (Parallel {asdf}) "flat_sharp.wav"

Print "Render flat and sharp works"
```

9.2.148 test/std_render_flatsharp.out

```
Render flat and sharp works
```

9.2.149 test/std_rhythms.nh

```
p = [1 5 6 7 1]
r = [1.0 2.0 1.0 0.5 2.2]
tr = init track{chords = p; durations = r}
out_r = Rhythms tr
if out_r == r then
  PrintEndline "Rhythms works"
else
  PrintEndline "Rhythms doesn't work"
```

9.2.150 test/std_rhythms.out

```
Rhythms works
```

9.2.151 test/std_scale.nh

```

(
large_scale = Scale (5@(-1)) (3@1)
manual_scale = [ 5@(-1) 6@(-1) 7@(-1) 1 2 3 4 5 6 7 1@1 2@1 3@1 ]

if large_scale == manual_scale then Print "Great" else Print "Not Great"
)
```

9.2.152 test/std_scale.out

```
Great
```

9.2.153 test/std_seq_tracks.nh

```
p1 = [1 5 6 7 1]
r1 = [1.0 2.0 1.0 0.5 2.2]
t1 = init track{chords = p1; durations = r1}
p2 = [1 2 3 4 5]
r2 = [1.0 2.0 3.0 4.0 5.0]
t2 = init track{chords = p2; durations = r2}
```

```

s = Sequential {t1 t2}
s_track = s$tracks
s_first_track = s_track.(0)
if s_first_track == {t1  t2} then
  PrintEndline "Sequential Tracks Work"
else
  PrintEndline "Sequential Tracks Don't Work"
```

9.2.154 test/std_seq_tracks.out

```
Sequential Tracks Work
```

9.2.155 test/std_simple_scale.nh

```
pitch_start = init pitch { rank = 1}
pitch_end = init pitch { rank = 7 }
out = Scale pitch_start pitch_end
test = [1 2 3 4 5 6 7]
if test == out then
  PrintEndline "Scale works"
else
  PrintEndline "Scale fails"
```

9.2.156 test/std_simple_scale.out

```
Scale works
```

9.2.157 test/std_track_extend.nh

```
tr = init track { chords = [ 4 1 2 3 4 ];           durations = [ 1.0 2.0 3.0 4.0 5.0 ] }

t1 = Extend 30.0 tr
t2 = init track { chords = [ 4 1 2 3 4 4 1 2 3 4 ]; durations = [ 1.0 2.0 3.0 4.0 5.0 1.0 2.0 3.0 4.0 5.0 ] }
PrintEndline (if t1 == t2 then "no padding :)" else "no padding :(")

t1 = Extend 31.0 tr
t2$chords = t2$chords . { (init chord) }; t2$durations = t2$durations . { 1.0 }
PrintEndline (if t1 == t2 then "padding :)" else "padding :(")
```

9.2.158 test/std_track_extend.out

```
no padding :)
padding :)
```

9.2.159 test/std_track_length.nh

```
p1 = [1 5 6 7 1]
r1 = [1.0 2.0 1.0 0.5 2.0]
t1 = init track{chords = p1; durations = r1}
p2 = [4 1 2 3 4]
r2 = [1.0 2.0 3.0 4.0 5.0]
t2 = init track{chords = p2; durations = r2}
l1 = Length t1
l2 = Length t2
PrintEndline(StringOfFloat(l1))
PrintEndline(StringOfFloat(l2))
```

9.2.160 test/std_track_length.out

```
6.500000
15.000000
```

9.2.161 test/std_track_octave.nh

```
p = [4 1 2 3 4]
ra = [1.0 2.0 3.0 4.0 5.0]
tr = init track{chords = p; durations = ra}
t3 = Octave 2 tr

y = t3$chords.(0)
z = y$pitches.(0)
a = z$octave

PrintEndline (StringOfInt (a))
```

9.2.162 test/std_track_octave.out

```
2
```

9.2.163 test/std_track_relative.nh

```
p = [4 1 2 3 4]
ra = [1.0 2.0 3.0 4.0 5.0]
tr = init track{chords = p; durations = ra}
p2 = [(1@1) 5 6 7 (1@1)]
r2 = [1.0 2.0 3.0 4.0 5.0]
t2 = init track{chords = p2; durations = r2}
t3 = Relative 4 tr

if t2 == t3 then
  PrintEndline "Track Relative Works"
else
  PrintEndline "Track Relative Doesn't Work"
```

9.2.164 test/std_track_relative.out

```
Track Relative Works
```

9.2.165 test/std_track_relative_neg.nh

```
p = [(4@1) (1@(-1)) (2@(-1)) (3@(-1)) (4@1)]
ra = [1.0 2.0 3.0 4.0 5.0]
tr = init track{chords = p; durations = ra}
p2 = [(1@3) 5 6 7 (1@3)]
r2 = [1.0 2.0 3.0 4.0 5.0]
t2 = init track{chords = p2; durations = r2}
t3 = Relative (-11) t2

if tr == t3 then
  PrintEndline "Track Relative Negative Works"
else
  PrintEndline "Track Relative Negative Doesn't Work"
```

9.2.166 test/std_track_relative_neg.out

```
Track Relative Negative Works
```

9.2.167 test/std_track_removeend.nh

```
p1 = [1 5 6 7 1]
r1 = [1.0 2.0 1.0 0.5 2.0]
t1 = r1:p1
p2 = [1 5 6]
r2 = [1.0 2.0 0.5]
t2 = r2:p2
t3 = RemoveEnd 3.0 t1

if t3 == t2 then
  PrintEndline "RemoveEnd Works"
else
  PrintEndline "RemoveEnd fails"
```

9.2.168 test/std_track_removeend.out

```
RemoveEnd Works
```

9.2.169 test/std_track_repeat.nh

```
p = [4 1 2 3 4]
r = [1.0 2.0 3.0 4.0 5.0]
tr = init track{chords = p; durations = r}
t1 = Repeat 2 tr
p2 = [4 1 2 3 4 4 1 2 3 4]
r2 = [1.0 2.0 3.0 4.0 5.0 1.0 2.0 3.0 4.0 5.0]
t2 = init track{chords = p2; durations = r2}

if t1 == t2 then
  PrintEndline "Track Repeat Works"
else
  PrintEndline "Track Repeat Doesn't Work"
```

9.2.170 test/std_track_repeat.out

```
Track Repeat Works
```

9.2.171 test/std_track_reverse.nh

```
p = [4 1 2 3 4]
r = [1.0 2.0 3.0 4.0 5.0]
tr = init track{chords = p; durations = r}
p2 = [4 3 2 1 4]
r2 = [5.0 4.0 3.0 2.0 1.0]
t2 = init track{chords = p2; durations = r2}
t3 = Reverse tr

if t2 == t3 then
  PrintEndline "Track Reverse Works"
else
  PrintEndline "Track Reverse Doesn't Work"
```

9.2.172 test/std_track_reverse.out

```
Track Reverse Works
```

9.2.173 test/std_track_start_with.nh

```
p1 = [1 5 6 7 1]
r1 = [1.0 2.0 1.0 0.5 2.2]
t1 = init track{chords = p1; durations = r1}
p2 = [3 2]
r2 = [1.0 0.5]
t2 = init track{chords = p2; durations = r2}

t3 = StartWith t2 t1
p_test = [3 2 5 6 7 1]
r_test = [1.0 0.5 1.5 1.0 0.5 2.2]
if((t3$chords == p_test) && (t3$durations == r_test))then
  PrintEndline "StartsWith Work"
else
  PrintEndline "StartsWith Don't Work"
```

9.2.174 test/std_track_start_with.out

```
StartsWith Work
```

9.2.175 test/throw_ifthenelse_nothrow.nh

```

if true then Print "yay" else throw "boo"
```

9.2.176 test/throw_ifthenelse_nothrow.out

```
yay
```

9.2.177 test/throw_simple.nh

```

throw "boo"
Print "yay"
```

9.2.178 test/throw_simple.out

```
boo
```

9.2.179 test/typedef.nh

```

type hi_hello = {
  x = 5
}
```

9.2.180 test/typedef.out

9.2.181 test/typedef_chained.nh

```

type first = {
  f = "hello"
}

type second = {
  s = init first
}

type third = {
  t = init second
}

x = init third

Print x$t$s$f
```

9.2.182 test/typedef_chained.out

```
hello
```

9.2.183 test/typedef_eq.nh

```
type big = {
  afdf = "hi"
  fdfd = "bye"
}

x = init big
y = init big { afdf = "hello"; fdfd = "goodbye" }
z = init big
if x == y then
  PrintEndline "x=y"
else
  PrintEndline "x!=y"
if x == z then
  PrintEndline "x=z"
else
  PrintEndline "x!=z"
if y != z then
  PrintEndline "y!=z"
else
  PrintEndline "y=z"
```

9.2.184 test/typedef_eq.out

```
x!=y
x=z
y!=z
```

9.2.185 test/typedef_init.nh

```

type override_init = {
  asdf = "hi"
}

x = init override_init { asdf = "hello" }
Print x$asdf
```

9.2.186 test/typedef_init.out

```
hello
```

9.2.187 test/typedef_large.nh

```

type big = {
  afdf = "hi"
  fdfd = "bye"
}

x = init big { afdf = "hello"; fdfd = "goodbye" }
Print x$afdf
Print "\n"
Print x$fdfd
```

9.2.188 test/typedef_large.out

```
hello
goodbye
```

9.2.189 test/typedef_print_fields.nh

```

type my_type = {
  a = "hi"
}


x = init my_type

Print x$a
```

9.2.190 test/typedef_print_fields.out

```
hi
```

9.2.191 test/typedef_simple.nh

```

type my_type = {
  a = 5
}

x = init my_type
Print (StringOfInt x$a)
```

9.2.192 test/typedef_simple.out

```
5
```