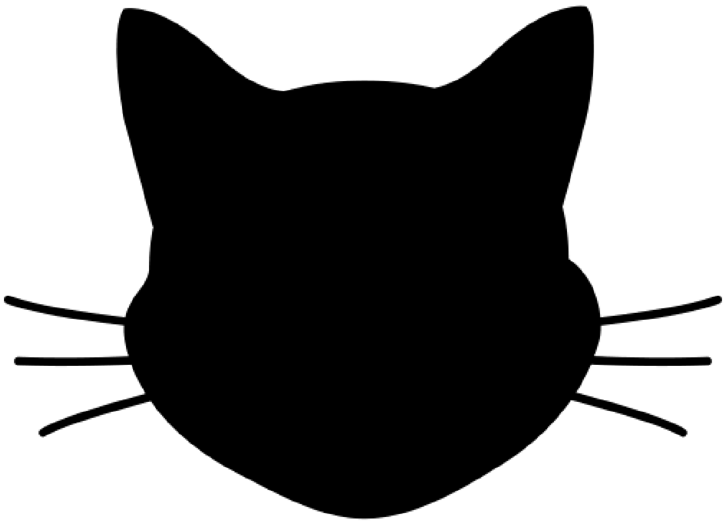
E-CATZ



Ethiopia Mengesha (em3353)
Chianna Cohen (clc2230)
Annie Sui (aqs2104)
Tim Vallancourt (tpv2106)

Introduction

The presence of electronic music is one of the definingtraits of contemporary music; however, to many musicians, it may feel daunting to make the step from traditional instruments to electronic music. E-CATZ, aims to help artists bridge this gap. E-CATZ is a statically typed imperative programming language created for musicians to easily and intuitively create music through code. With E-CATZ, artists will be able to compose music without the limitations of physicalinstruments, allowing far more creative freedom. E-CATZ aims to strike a balance between being intuitive for the classically trained musician and facilitating music creation that is uniquely digital. Thus it has Java-like syntax, but it is built around the basic unit of a note.

E-CATZ draws reserved words from a musician's vocabulary and includes built in operations specific to music composition. Musicians are able to string these notes into an array to create simple melodies out of a sequence of notes. They can create more complex soundsthrough 2D arrays, which allow the musician to string together chords. It includes a standard library to aid the musician in creating music through array concatenation and octave shifts. We also prioritized creating built-in functions for randomness, an essential and unique feature of the digital art space.

The language is designed to allow the creation of LilyPond files, a versatile form of music notation,though our built in write functions. The LilyPond files can be used to create both MIDI files, the industry standard for electronic musicians or sheet music. Ultimately, E-CATZ provides users an accessible way to create their own musical projects and experiment with sound.

Tutorial

Environment Setup

Before using E-CATZ, it is necessary to set up your development environment. Instead of installing all the packages manually, we recommend using this premade Docker image.

Install Docker on Ubuntu:

```
apt install docker.io
```

Move into the E-CATZ directory

```
cd ecatz
```

Run Docker:

```
docker run --rm -it -v `pwd`:/home/ecatz -w=/home/ecatz columbiasedwards/plt
```

And you should be good to go!

Compiling and Running

To build the E-CATZ compiler:

```
make
```

To see the AST/SAST/LLVM of a program:

```
./ecatz-native [-a|-s|-l] [program.catz]
```

Note: if you would like to do any of the above actionswith a program that uses the include keyword, it is necessary to first preprocess your E-CATZ code:

```
./preprocessor [program.catz] > [newname.catz]
./ecats-native [options] [newname.catz]
```

To preprocess and execute your program:

```
./run-ecatz.sh [program.catz]
```

Sample Programs

Hello World

Writing a program in E-CATZ is very simple. Here is Hello World

```
prints("Hello World!");
```

Loops, Math, Branches, Functions

Here is another program that demonstrates some of the common features that E-CATZ implements. It doesn't do anything meaningful.

```
def addFive(int x) {
  int y = x + 5;
  return y;
}

int result = 0;
int other;
for (int i = 0; i < 10; i = i + 1) {
  result = addFive(result);
  other = result * 4;
  if (other > 25) {
    prints("big number, huh");
  }
}
```

Language Features

In this section, we will go over how to use some of the more interesting features of E-CATZ.

Notes

Notes are the building blocks of music. In E-CATZ, they are effectively structs of two values: a pitch and a rhythm.

A pitch literal is made up of 2 to 3 parts: a letter representing a note (uppercase letters A-G), followed by optionally a flat or sharp symbols (@ or #), and then ended with the octave number (a single digit number, 0-9).

A rhythm literal is two lowercase letters that are directly connected to a basic rhythm structure. The options are ts, st, et, qr, hf, and wh which correspond to 32nd notes, 16th, 8th, quarter, half, and whole notes.

There are two ways to build a Note.

Use the note constructor and specify a pitch and rhythm:

```
Note myNote = new Note(C#5, hf);
```

Or just use a pitch literal which will create a note with a default rhythm (quarter note).

```
Note myNote = B3;
```

The pitch and rhythm of a Note can be accessed and modified using the dot (.) operator.

```
Note x = new Note(F#3, wh);
Note y = new Note(x.pitch, hf);
x.rhythm = ts;
```

Arrays

Arrays are ordered collections of items. They can be constructed two ways:

```
int[] x = new int[10]; /* empty array of length 10 */
int[] y = [3, 5, 6, 7];  /* array literal with 4 values */
```

The length of an array can be found with the dot operator.

```
x.length; /* 10 */
y.length; /* 4 */
```

We also support nested arrays:

```
int[][] a = new int[][5]; /* array of 5 empty int arrays */
int[][] b = [[3, 5, 7], [1, 2]]; /* array of 2 arrays of varied length */
```

Elements can be accessed and rewritten using bracket notation.

```
y[2]; /* 6 */
b[1][0]; /* 1 */
x[0] = b[0][2]; /* reassign first value in x to 7 */
```

Write

The write function takes your created musical structures in E-CATZ and writes them out into a LilyPond (.ly) file. There are two write functions: write and writeN. Write takes in an array of notes and a string, specifying output file name. Calling the write function will write all the notes in the array sequentially.

This will produce a LilyPond file of 4 quarter notes played sequentially.

```
Note[] output = [D3, E3, F#3, G3];
write(output, "4notes");
```

WriteN takes in an array of arrays of notes instead.Each subarray will be written as a group of notes to be played simultaneously, and subsequent subarrays will be played afterwards.

This will produce a LilyPond file of a C major chord followed by a G major chord.

```
Note[][] output = [[C3, E3, G3], [G3, B3, D4]];
write(output, "2chords");
```

LilyPond and MIDI

The write function outputs LilyPond files. How do you actually use LilyPond files? Follow the instructions at http://lilypond.org/download.html to download and install LilyPond. Once installed you can open a .ly file produced by an E-CATZ program in LilyPond. From there, you can click Compile > Typeset file. This will compile the LilyPond file and produce a MIDI file in the same folder as the LilyPond file.

MIDI files are the industry standard for communicating musical sequences digitally. They can be opened in a wide variety of programs, such as Ableton Live, Logic, GarageBand, FL Studio, and more. From there, you can use these MIDI tracks however you look in your music production. The simplest way to listen to your MIDI file is to open it in VLC Media Player, which can be downloaded and installed using the instructions at that link. Open the MIDI file in VLC and it will use default instruments to play the file you produced.

Reference Manual

Lexical Conventions

In the E-CATZ language, tokens can be separated into the following categories: identifiers, keywords, constants, expression operators, and other separators. Whitespace and comments are ignored except in the case they are contained in a string. Whitespace is required to separate identifiers or constants that would otherwise be adjacent to each other.

Comments and whitespace

The characters /* introduce a comment, which terminates with the characters */. Anything between these symbols is ignored by the compiler; whitespace characters are also ignored.

Identifiers

An identifier is defined as the name of a variable. It must be a sequence of letters, digits, and underscores. The first character must be lowercase and alphabetic. After the first character, it can be any letter, digit, or an underscore.

Operators

The following symbols are reserved operators in E-CATZ:

`+  -  *  /  %  ==  !=  <  <=  >  >=  &&  ||  !`

Their functionality will be explained later.

Keywords

Keywords are a specific collection of characters which are not to be used as identifiers. Keywords include the name of primitive types, builtin functions, built-in expressions, rhythm, boolean, and Note values, and the words new and def. The following identifiers are reserved for use as keywords, and may not be used otherwise:

Types	int, bool, float, Note, String, void
Builtin functions	print, printf, prints, randInt, seed, write, writeN
Rhythm values	ts, st, et, qr, hf, wh
Boolean values	true, false
Pitch values	[A-G](#|@)?[0-9]
Struct fields	pitch, rhythm, length
Other keywords	if, else, for, while, return, new, def, include

Constants and Literals

There are eight types of constants or literals supported in the E-CATZ language: ints, bools, floats, Notes, pitches, rhythms, Strings, and arrays.

Integer constants are a sequence of digits.

Boolean literals are the keywords true and false.

Floats are decimal numbers that must be written with periods. They can be either digits followed by a period, a period followed by a digit, or a period with digits on both sides (so .5, 1.5, and 1. are all valid floats)

Note literals and pitch literals are both sequences of two or three characters represented by the regular expression [A-G](#|@)?[0-9]. This represents a pitch and an octave. E-CATZ knows contextually when to treat this expression as a note literal and when to treat it as a pitch literal.

Rhythms are two characters written together from the predefined list seen in the above table.

Strings are a sequence of ASCII characters surrounded by double quotes.

Array literals are a sequence of expressions separated by commas between two brackets: [expr1, expr2,...]. Each expression must be of the same type. These expressions are now collected together in a single structure. An array literal must contain at least one expression. An empty array literal [] is an error.

Statements & Expressions

Expressions

The subsections are listed in their order of precedence. The convention is that higher precedence operators are performed first.

Accessible

At the highest precedence are two expressions that access stored information.

Identifier

The type of an identifier is determined by its declaration. We define variable identifiers to be identifiers that do not precede the '()' signs.

Bracket Access

Accessible expressions of the array type can have their contents accessed by using bracket notation.

accessible[expression]

The expression must be of type integer. It returns an element from the arrays, so is therefore the same type as whatever the array contains.

Dot Access

Both arrays and notes have properties that can be accessed using the a dot (.) These are all explicitly defined. Arrays have a property length of type int. Notes have two properties, pitches and rhythms, which are of type pitch and rhythm respectively.

accessible.rhythm

accessible.pitch
accessible.length

Function Calls

Functions are called like this:

ID(*args_opt*)

Where *args_opt* is a comma separated list of expressions that the function requires to take in as input. The number of expressions in *args_opt* must correspond with the number of parameters specified in the function definition. The type of a function call is the same as the return type of the function.

Constructors

There are two constructors which create each of the objects in E-CATZ, Note and Array.

The constructor of Note is as follows. The first expression must be a pitch literal and the second expression must be a rhythm literal. They must both be present. It returns a value of type Note.

new Note(expression1, expression2)

The constructor of an Array is as follows. Type must be the name of one of the types in E-CATZ, including another array. Expression should evaluate to an integer that will specify the length of the array.

new typ[expression]

(Expression)

The type and value of an expression are unaffected by the presence of parentheses.

Operators

Unary Operators

! expression

The result of the logical negation operator ! is true if the value of the expression is false, and false if the value of the expression is true. Thetype of the result is boolean. This operator is only applicable to booleans.

- expression

The result of the unary minus operator is the negative of the expression, and has the same type. The type of the expression must be int or float.

Multiplicative Operators

Multiplicative operators *, /, and % group left-to-right.

*expression * expression*
The binary operator * signifies multiplication. Both operands are allowed to be type int or float. If both operands are int, the result is an int. If both operands are float, the result is a float.

expression / expression
The binary / operator indicates division. Both operands must be type int or float, and the result is an int or float.

expression % expression
The binary % operator yields the remainder from the division of the first expression by the second. Both operands must be of type int and the result is an int.

Additive Operators

Additive operators +, - group left-to-right.
expression + expression
The result is the sum of both expressions. Both operands must be of the same type. The operand types allowed are int or float. For ints and floats, the result is the sum of the two numbers.

expression - expression
The result is the difference of both expressions. Both operands must be type int or float.

Relational Operators

expression < expression

expression > expression

expression <= expression

expression >= expression

The operators <, >, <=, and >= all yield false if the specified relation is false and true if it is true. The operands must be both type int or float.

Equality Operators

expression == expression

expression != expression

The equality operators have lower precedence compared to the relational operators. The == operator returns true if both operands have the same value. The != operator returns false if otherwise. Operands must be of the same type. They can be ints, floats, pitches, or rhythms.

Logical Operators

All logical operators listed are left associative.

expression && expression: Returns true if both operands are true, and false otherwise.

expression || *expression:* Returns true if one of theoperands is true, and false otherwise.

Assignment Operators

Assignment is right-associative and returns the assigned value.

accessible = expression

This expression assigns the result of expression to the expression corresponding to accessible.

There are also two special expressions corresponding to the assignment of fields of a note object.

accessible.pitch = expression

accessible.rhythm = expression

These expressions assign the result of expression to either the field pitch or rhythm of the expression corresponding to accessible (which must be a note).

Statements

Bind Statement

Bind statements bind an identifier to be a specific type. Once an id is bound to one type, it cannot be bound to another type, or re-bound. Bind statements are of the form:

typ id ;

Bind Assign Statement

Bind Assign statements are similar to bind statements, but they additionally assign an expression to be the value of the id, as it's being assigned.

typ id = expression ;

Expression Statement

Our statements will primarily take the following form:

expression;

Expression statements are used for assignments or function calls.

Conditional Statement

The conditional statement has the basic forms:

if *(bool_expression) statement*

if *(bool_expression) statement else statement*

In both cases, the expression is evaluated. If the expression evaluates to true, the first statement is executed. In the second form, if the expression evaluates to false, the second statement is executed. The statements are typically of the form of a block,that is a list of statements surrounded by curly braces.

While Statement

The while statement has the form

while (bool_expression) statement

The substatement is executed repeatedly so long as the value of the expression remains true. The test takes place before each execution of the statement.

For Statement

The for statement has the form

for (expression1; bool_expression; expression3) statement

The first expression initializes the loop, the second specifies the conditions that must be satisfied before each iteration, and the third indicates an action taken after each loop, typically an increment or decrement of the value initialized in the first expression. The statement is executed repeatedly so long as the value of the expression remains true.

An equivalent statement in the form of a while loop can be written as follows:

expression1;

while (bool_expression) {

    statement;

```
        expression3;
    }
```

Return Statement

The return statement enables a function to return to its caller, and takes either of the two following forms:

```
return;

return expression;
```

The first case does not return a value. The second case returns the value of the expression to the function caller. The type of the expression returned must correspond with the type declared in the function header.

Block Statement

The block statement groups statements together for the purpose of defining functions or the bodies of loops and branches. It looks like this:

```
{ statement_list }
```

Where a statement_list is zero or more statements.

Function Declarations

Other than statements, code can also be in the form of a function declaration. Functions are callable blocks of code. By specifying the return type, identifier, and list of zero or more parameters, statements will be grouped such that they can be called at other points in the code.

Declaration of a function looks like this:

```
def typ ID(parameters) {
    statement_list
}
```

Include

The include statement takes the form:

```
# include "filename"
```

This call indicates the given file is to be copied into the current file so that its contents can be accessed. This is necessary due to the E-CATZ language's reliance on files in the standard library. The include statement is handled by the C preprocessor which facilitates the inclusion of the named file. The contents of the included file are copied into the current file which allows past code to be reused and built upon.

Scope Rules

The source text of the program may be kept in several files. Use the include keyword, as explained above to use the text of a different file. Communication among the functions of a program may be carried out through explicit calls. That is, if a function exists in one file, like a standard library, it can be directly called in any other file, without needing to be rewritten or specifically initialized.

Lexical Scope:
The scope of an identifier exists from when it is declared to the following right braces or the end of the file. That is to say, if an identifier is first declared inside a function definition, a loop, or an if/else statement, it will only exist inside that context. There are no global identifiers. Identifiers declared inside the "main" scope (i.e. not inside a function) will not be usable inside other functions, even if they are also defined within that main scope.

An identifier cannot be re-declared; this is an error. It can however be reassigned to another expression of the same type.

Builtin Functions

The E-CATZ language contains 7 builtin functions that all are references to functions written in C.

Print Functions

There are three print functions available to users. They are print, printf, and prints, which correspond to printing an integer, a float, and a string respectively. They all take a single argument – an expression of the required type. These call the C print function, and the value is printed to standard out.

Write Functions

There are two functions related to writing out to LilyPond files: write and writeN. Write takes two arguments, an array of Notes and a String. Write takes each of the notes in the array and writes them out sequentially to a LilyPond file of the name specified by the string. WriteN takes two arguments, an array of arrays of Notes and a String. Write takes each of the subarrays and writes them out sequentially to a LilyPond file. Each subarray is written by taking all of the Notes in that subarray and writing them to occur simultaneously in the LilyPond file.

Random Functions

There are two functions related to random number generation. The first is seed() which takes no arguments. This calls the C function srand() with the argument time(NULL), allowing for a random seed to be generated.

The seed function must be called before using the other random function, randInt. RandInt takes two ints as arguments, one specifying the lower bounds and the other specifying the upper bounds and returns a random integer between those bounds.

Standard Library

The E-CATZ language comes with a standard library that contains functions related to Note-int equivalency, note addition and subtraction, and array concatenation.

Note-Int Equivalency

There are two functions responsible for this equivalency: noteToInt() and intToNote(). NoteToInt takes a note as an argument and returns an integer that is effectively equivalent in E-CATZ. intToNote takes an integer as an argument and returns its Note counterpart. This equivalency is merely defined contextually within E-CATZ and works because it is consistent with itself. These functions are mostly useful for being able to do addition and subtraction on notes later.

Note Addition and Subtraction

addToNote() takes two arguments: a note and an integer, which can be positive or negative and returns a new Note. The integer specifies how many semitones the new Note should be higher or lower than the input Note. For example, addToNote(C4, 1) would return C#4 because it is one semitone higher.

octaveChange() takes two arguments: a note and an integer, which can be positive or negative and returns a new Note. The integer specifies how many octaves the new Note should be higher or lower than the input Note. For example, addToNote(C4, 2) would return C6 because it is two octaves higher.

Array Concatenation

Presently, there are two concatenation functions defined in the standard library, one for concatenating arrays of Notes and one for concatenating arrays of arraysof Notes. It would also be possible to define more concatenation functions for other types, but these were the only two made so far. For each function, you pass in two arguments which are both arrays of the propertype. The functions then return those two arrays glued together.

Project Plan

Planning Process

To ensure that we kept ourselves on track, we set a weekly team meeting time on Sundays, met with our TA Harry on Wednesdays, and created checkpoints for each step of the project by the end of every meeting. Through consulting Harry and Prof. Edwards, we were able to gauge how much progress we had made relative to our goals. We divided larger goals into smaller subgoals and distributed the work to each team member depending on each individual's level of expertise.

Development Process

We first tackled the less tedious components that would fulfill the basic functionality requirements of our programming language. This included parsing, lexing, and semantics checking. To implement basic components such as if-else statements and for-loops, we referenced MicroC and then added new parts that specifically catered to our own language. We then developed the code generator, which was more time consuming but enabled us to achieve a basic running program.

Testing Process

We implemented tests as we completed each individual component of our language. When one part of the compiler was built, a team member would write pass/fail unit tests for that particular component. More specific details on how we created our test suite will be outlined in an upcoming section.

Team Responsibilities

Note that there were many overlapping roles among team members, as we often split up into groups of two to work on parts of the project over time.

Team Member	Responsibilities
Tim Vallancourt (Language Guru)	Compiler front and back end, Semantic Checking, Code generation
Annie Sui (Project Manager)	Compiler front and back end, Semantic checking, Documentation
Ethiopia Mengesha (System Architect)	Compiler front and back end, Semantic Checking, C libraries
Chianna Cohen (Tester)	Compiler front and back end, Semantic checking, Automated testing suite

Project Timeline

Date	Milestone
February 3	Submitted Project Proposal
February 14	Created E-CATZ repository, first commit
February 24	Submitted Language Reference Manual
March 15	Hello World Parser and Scanner complete
March 23	Hello World Semantics and Code Generation complete
March 24	Hello World runs
April 12	Write function complete
April 13	Parser and Scanner complete
April 18	Code Generation complete
April 26	Presented project and submitted Final Report

Software Development Environment

The following programming and development environments were used in the construction of our language:

- Programming language for building compiler: Ocaml version 4.11.1. The scanner and parser were compiled using Ocamlyacc and Ocamllex extensions.
- Development environments: Docker, vim, Visual Studio Code

Project Log

See appendix C.

Programming Style Guide

The conventions that were used to build our language are based on standard Ocaml formatting guidelines. To ensure better readability and consistency across files, our team adhered to the following rules:

- Only 2-space and 4-space tabs are used in each program.
- When naming variables, use snake_case instead of camelCase convention.

Comments and documentation:

- Each section of Ocaml code is preceded by multiline comments that describe the purpose of the lines to follow.

Language Evolution

The Beginning

There were hopes, dreams, idealism, optimism, and, most of all, naïvité. Before we had any sense of what it would actually entail take to build a programming language (what would be easy? Difficult? Impossible?), we had a proposal with our long list of features. The main idea was that we wanted to allow users to create notes, put them into data structures, and be able to write the results into MIDI files that could be played and listened to. We hoped to have many common language features: loops, branches, other control flow statements, functions, variable declarations, and various binary and unary operators. We wanted to have note objects that would contain data about notes and dot access to those notes to allow users to work with the individual components. We also planned to have arrays (and slices) and special classes of arrays called sequences and harmonies, that would impart certain musical rules upon the notes they contained. We also wanted to include additional note specific operators that would allow uses to do more interesting things with them, like modifying their pitches. In addition to this, there were a few built-in functions that would exist: random number generation, writing midi files, and reading midi files. Lastly, we proposed automatic garbage collection, which we quickly found out would be far too difficult and not worth our while.

The Transformation

MIDI -> LilyPond

In our initial proposal, we hoped that our program would directly write MIDI files. That is, when looking at a sequence of notes, it would look at each note and write out the series of bytes that would correspond to it. The MIDI information of a note includes its pitch, time since last input, and velocity. Our function that would write the MIDI files would need to look at the information in our Note structures and determine what to write. Particularly difficult would be looking at the standard rhythm values we planned to use (quarter note, half note, etc.) and translating them into what the MIDI file actually wanted which was a time value in milliseconds since the last note. Additionally, it was also just very messy to be writing bytes directly as we could not directly read any of our own outputs.

After a conversation with Professor Mark Santolucito, we took his recommendation to write to a file format other than MIDI. One of his suggestions was LilyPond, a text-based music notation software. With this program, users can write in plain text and LilyPond will compile it into both a visual musical score and into a MIDI file. Because LilyPond files are human readable and use standard rhythm notation, they are much easier to work with and debug. This also still allowed us to ultimately output a MIDI file, just with an intermediary. Before, it seemed like we would have to dedicate significant resources to figuring out how to write MIDI files in C (instead of actually working on our language in OCaml). Now, we could push that work to someone else and get the same results.

Note Representation

With moving to LilyPond also came simplifying our note representation. Initially, notes were proposed to be a struct of four parts: pitch, rhythm, velocity, and MIDI number. MIDI number and pitch were to be linked, with pitch just being the human-readable version of the MIDI number. As we were no longer directly writing MIDI events, we could eliminate the MIDI number as it was redundant information. We also chose to eliminate velocity because, though necessary when writing a MIDI event, it is not as interesting as pitch and rhythm as most of the time it is equivalent to volume. Our notes went from being structs of 2 strings and 2 int, to just being a struct of 2 strings.

Harmony & Sequence Types -> Generic Nested Arrays

We also originally proposed special data types called harmonies and sequences. Harmonies would effectively be arrays of notes that would be played simultaneously, like a chord. Sequences would be arrays of either notes (or harmonies) that would be played sequentially. It became apparent that these were merely wrapper classes that were adding an unnecessary layer of complexity to our language. Instead, we decided to implement generic arrays and also make it possible to also feature nested arrays. Effectively, a sequence would be a simple array of notes and a harmony would be an array of arrays of notes. Within the language of E-CATZ, they are just these arrays. However, you would then call two different write functions: write and writeN. Write takes an array of notes and writes them as a sequence, so each note just is played after the other. WriteN (short for write nested) takes in an array of an array of notes and writes them as a sequence of harmonies, so each sub-array is written as a chord, where the notes are played simultaneously. This gave us the functionality that we desired without unnecessary syntax. In some documentation, we still refer to them as harmonies and sequences as they became too ingrained in our minds.

The Culling

As the semester came closer to the end, it became necessary to eliminate features that we had initially proposed (and would have liked to have), but were not strictly necessary to the functionality of our programming language. Enhanced for loops (for iterating over the elements of an array) and combined operator assignments (+=, -=, *=, .=) were the first to go because the same functions could already be performed in E-CATZ, just with different syntax. Along the same lines, array slicing was also removed as a potential feature because, while nice to have, you could ultimately do most of the same things with a for loop that only iterated over a certain part of the array.

A number of operators (on notes and arrays) that we had planned to implement also had to be eliminated in favor of other features. Some of the operators were difficult because our representation of a note had changed, as outlined above. We originally planned certain unary operators on notes that allowed one to change the pitch of the note either by a semitone or octave. Because notes were now just strings and not ints, this would not be feasible without major reworking. There were also plans to be able to directly add values to notes (and raise or lower the pitch) with a plus operator. Array concatenation would also be possible with + and * but this proved difficult (and easier to just do in the standard library, as will be explained later).

Lastly, we decided not to implement a read function which would work as the mirror of our write function: it would take in a LilyPond file and turn it into nested arrays that we could use in our language. We made this decision mainly for two reasons: (1) LilyPond files can be in a variety of styles and have very many headers, so accounting for all of those would be cumbersome, and (2) making the read function would really be an exercise in writing C code that can parse another file and not actually have anything to do with making our programming language work.

As sad as it was, the culling was a necessary step in the evolution of our language.

The Standard Library

And yet, some victims of the culling were reborn in the standard library. When attempting to implement some of the culled features in LLVM, it became apparent that things that would be very complicated in that context would become trivial if we just wrote it in E-CATZ instead. Though we only implemented a subset of these, it would be possible to add a number of interesting features to the standard library.

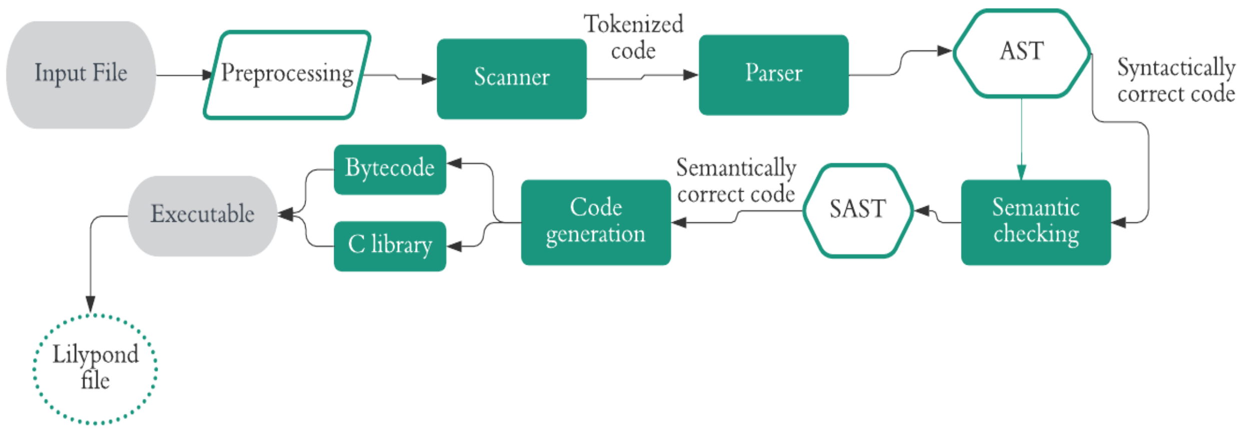
Replacing an integer representation of notes is a lookup table of pitches that allows one to convert a pitch to an integer equivalent. This then allowed us to implement something like our proposed note addition, subtraction, and octave change features. A user could call these functions and it would search the equivalency table and allow you to change pitches by integer values. Array concatenation also moved to the standard library as it became obvious that it was very easy to write this in E-CATZ and very annoying in LLVM. Right now, there are array concatenation functions for Note arrays and nested Note arrays. It would be trivial to add the function for other types.

Though it only has these few functions, the standard library became a hopeful place where it was possible to write relatively simple code that would make our programming language much more powerful. See stdlib.catz in Appendix A for source code.

The Finale

In the end, despite many of the changes our language underwent during its creation, it still satisfied our initial vision: users can create music in our language and ultimately get a MIDI file from it. Though we lost features along the way, we figured out many workarounds and were able to implement the main features that we wanted. Many details of our language changed, but the bones remained the same. Overall, our language evolved as it should and went from over-optimistic idea to an implemented reality.

Translator Architecture



*see Appendix A for compiler source files

Preprocessing:

Before all else, a preprocessing shell script is executed on the input file. The file is first run through the C language preprocessor which copies included files and the code contained in the input file into an intermediary file. Then, extraneous lines created by the C preprocessor are removed to prepare the file for the scanner.

Source Code:

```
#include "test-if.catz"
#include "test-math.catz"
math(6);
math(10);
```

Intermediate Code:

```
# 1 "./tests/test-include.catz"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "./tests/test-include.catz"
# 1 "./tests/test-if.catz" 1
```

```
def void math (int x){
if (x<7){
    print (1);
}
else if (x==7){
    print(8);
}
else{
    print(20);
}
}
math(7);
# 2 "./tests/test-include.catz" 2
# 1 "./tests/test-math.catz" 1
print(4 + 4);
print(6 / 2);
print(5 * 4);
print(6 - 3);
print(3 % 2);
printf(4.2 + 4.2);
printf(3.1 - 1.2);
printf(4.1 * 5.);
printf(9.9 / 3.);
# 3 "./tests/test-include.catz" 2
math(6);
math(10);
```

Preprocessed Code:

```
/* from test-if */
def void math (int x){
if (x<7){
    print (1);
}
else if (x==7){
    print(8);
}
else{
    print(20);
}
}
math(7);
/* from test-math */
print(4 + 4);
print(6 / 2);
print(5 * 4);
print(6 - 3);
print(3 % 2);
printf(4.2 + 4.2);
printf(3.1 - 1.2);
printf(4.1 * 5.);
printf(9.9 / 3.);
/* new calls made in test-include */
math(6);
math(10);
```

Scanner :

The preprocessed file is passed into the scanner for lexical analysis. Here, each character is scanned and compared to the recognized character(s) defined in scanner.ml. Recognized characters are converted into tokens and unrecognized characters cause an illegal character error.

Parser and AST:

The tokenized code produced by the scanner is passed into the parser which recognizes how the tokens fit together within larger syntactic structures. From context tokens are recognized by the parser as types, statements, expressions, function declarations, and other similarly complex groupings. In conjunction with ast.ml the parser attempts to create an Abstract Syntax Tree and checks for syntactic correctness.

Semantic Checking:

In semant.ml the Abstract Syntax Tree is semantically checked as per the language constraints defined in semant.ml and sast.ml. Semantic checking accounts for errors like expected type(s), argument lengths, instantiation, and initialization. A semantically correct syntax tree is converted into a Semantically-checked Abstract Syntax Tree in this step.

Code Generation:

The SAST generated by semant.ml and the AST generated by the parser are evaluated by codegen.ml, the last step in the compilation process. Code Generation represents the grammar, syntax, and semantics of E-CATZ in LLVM by translating the desired functionality of code written in our language into the appropriate LLVM instructions.

Utils in C

The translation to LilyPond is facilitated by structs and functions written in the C language. The functions *write*, *writeN*, *randInt*, and *seed* as well as structs for Note, Sequences, and Harmonies are defined in utils.c. Based on the output of codegen, utils.c creates a LilyPond file and populates it with the correct LilyPond code.

Test Plan and Scripts

Testing Plan

Throughout the process of creating the compiler, we wrote unit tests for each feature of our code individually. This includes tests for basic arithmetic, loops, conditionals, include, and creating basic functions. We also created particularly extensive testing for Array and Notes to verify their features worked correctly. For each feature that worked, we included fail tests where the feature was used incorrectly to make sure semant.ml was working correctly. We only included compile time errors in our testing suite, so no segmentation fault errors, index out of bounds errors, or other runtime errors. We also did not include any parsing errors.

After the entire compiler was built we created some integration tests to ensure all of the parts were working together properly in a more complex program. This includes a program creating a sample of Never Gonna Give You Up, a program using serialism, a method of composition based sequences of notes, and a program iterating on the C major blues scale.

Automating Testing

Testing was automated using a modified version of the shell script from microC. There are three types of tests it automatically runs: success tests, fail tests, and LilyPond tests, which are denoted by the prefixes test-*, fail-*, and write-* respectively.. For the success tests, the script pipes standard out into a .out file and for the fail tests, it pipes the result in standard error into a .err file. In order to test our Note features, the write function, and writeN function, we added a special method to the shell script that tested the programs that resulted in LilyPond files. These tests automatically created .ly files that we used for evaluation. The script automatically runs each test and compares the output with the expected output file, error file, and LilyPond files, printing the file name it is testing along with OK if it matches or FAILED if it does not.

The tests that used our built-in random function were tested manually, due to their unpredictable nature. Our integrations tests are included in this category and are not automated, except for the blues scale piece called write-integration.catz, due to their use of rand-int and creation of multiple LilyPond files. Instead they were tested manually.

The output of our test script is as follows*:

```
test-arrayElementAssign...OK
test-arrayLength...OK
test-arrayNotes...OK
test-bind...OK
test-bindAssign...OK
test-comment...OK
test-createArray...OK
test-for...OK
test-helloWorld...OK
test-if...OK
test-include...OK
test-math...OK
test-neg...OK
test-nestedArray...OK
test-nestedArrayConstructor...OK
test-not...OK
test-pitchEquality...OK
test-precedence...OK
test-printingAnArray...OK
test-return...OK
test-rhythmArray...OK
test-sequentialAssign...OK
test-string...OK
test-voidReturn...OK
test-while...OK
fail-badBinop...OK
fail-conditionalType...OK
fail-doubleBind...OK
fail-illegalArrayLiteral...OK
fail-illegalAssignment...OK
fail-illegalBinary...OK
fail-illegalBracketAccess...OK
fail-illegalUnary...OK
fail-incorrectParameters...OK
fail-lengthBadType...OK
```

```
fail-pitchAccess...OK
fail-printPitchAndRhythm...OK
fail-rebindVariable...OK
fail-redefineFunction...OK
fail-undefinedFunction...OK
fail-undefinedVariable...OK
fail-voidBind...OK
fail-voidBindAssign...OK
fail-voidBindParameter...OK
write-2DNoteArrWithSingleNote...OK
write-2DNoteArray...OK
write-NoteArray...OK
write-integration...OK
write-noteAccess...OK
write-noteAssign...OK
write-noteConstructor...OK
write-reassignNoteFromPitch...OK
write-sequentialNoteAssign...OK
```

*Test for random int feature, full serialism program, and rick-roll program not included as they were tested manually

See Appendix B for the testall.sh script that runs the tests and all of the test files. See Appendix D for our full programs. .out, .err, and Lilypond files not included.

Conclusions

The objective of E-CATZ was to provide users with a way to experiment with music in an accessible and intuitive way through code. Although we decided to exclude some features that we had initially proposed, we developed a language that integrates the fundamental components of music composition and succeeded in meeting our overall end goal.

We learned that not only was setting deadlines important, but also the process of actually meeting those deadlines and figuring out what to do when our subgoals weren't accomplished in the timeframe that we anticipated. Creating a priority list helped us identify which tasks were independent of each other, which tasks were more difficult than others to implement, and features which would lead to more possible features once completed. For instance, creating arrays was one of the more difficult tasks, since we could not directly consult MicroC for hints on array implementation and had to implement arrays in a way that catered to our language specifically. However, once we had one-dimensional arrays working, we were then able to implement harmonies and sequences to provide extended user functionality. Team communication and adrenaline also helped us keep each other on track—whenever one of us felt the weight of a deadline, so did the rest of us.

Doing a team project in which you develop a programming language from scratch during the COVID-19 pandemic sounds nearly impossible when you first hear it, but with each of our team members supporting each other, we managed to hold up the fort. For those of you planning to take this class next semester: find a good team to work with and start early, you won't regret it!

Appendices

Appendix A:

preprocessor

```
#!/bin/bash
cpp -o - $@ > temp1.catz
sed '/^#/d' temp1.catz
```

scanner.mll

```
{ open Parser }

let digit = ['0' - '9']
let digits = digit+

rule tokenize = parse
  [' ' '\t' '\r' '\n'] { tokenize lexbuf }
| "/*" { comment lexbuf }
(* Operations *)
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| '%' { MOD }
(* Braces/Brakets/Paren etc *)
| '(' { LPAREN }
| ')' { RPAREN }
| '[' { LBRACK }
| ']' { RBRACK }
| ';' { SEMIC }
| ',' { COMMA }
| '{' { LBRACE }
| '}' { RBRACE }
| '=' { ASSIGN }
| '.' { DOT }
(* Keywords *)
| "def" { DEF }
| "return" { RETURN }
| "while" { WHILE }
| "for" { FOR }
| "new" { NEW }
| "rhythm" { RHYTHM }
| "pitch"  { PITCH }
| "length" { LENGTH }
(* Types *)
| "void" { VOID }
| "bool" { BOOL }
| "int" { INT }
| "Note" { NOTE }
| "float" { FLOAT }
| "String" { STRING }
(* Loops/Statements *)
| "if"     { IF }
| "else"   { ELSE }
(* Comparisons *)
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
(* Unary Operators *)
| "!" {NOT}
(* | ">>" {SEMUP}
| "<<" {SEMDOWN}
| "++" {OCTUP}
| "--" {OCTDOWN} *)
(* Literals *)
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| ['A'-'G']['@' '#']?digit as pitch { PLIT(pitch) }
| 't''s' | 's''t' | 'e''t' | 'q''r' | 'h''f' | 'w''h' as lit {RLIT(lit)}
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*  as id { ID(id) }
| (digits '.' digit*) | ('.' digits) as lxm { FLIT(lxm) }
| '"' {str (Buffer.create 16) lexbuf}
| digits as lxm { LITERAL(int_of_string lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
```

```
and comment = parse
  "*/" { tokenize lexbuf }
 | _ { comment lexbuf }

and str buf = parse
'"' { STRLITERAL( Buffer.contents buf) }
| _ {Buffer.add_string buf (Lexing.lexeme lexbuf); str buf lexbuf}
```

parser.mly

```
%{ open Ast %}

/* punctuation tokens */
%token LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK SEMIC COMMA ASSIGN DOT
/* arithmentic operator tokens */
%token PLUS MINUS TIMES DIVIDE MOD
/* keyword tokens*/
%token DEF RETURN WHILE FOR NEW RHYTHM PITCH LENGTH IF ELSE
/* comparison tokens*/
%token EQ NEQ LT LEQ GT GEQ AND OR
/* data types tokens */
%token BOOL INT FLOAT NOTE STRING VOID
/* unary ops */
%token NOT

%token <string> PLIT ID FLIT STRLITERAL RLIT
%token <int> LITERAL
%token <bool> BLIT
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%left SEMIC
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT GEQ LEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
```

```
%right NOT

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { ([], [])               }
 | decls stmt { (($2 :: fst $1), snd $1) }
 | decls fdecl { (fst $1, ($2 :: snd $1)) }

fdecl:
   DEF typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { typ = $2;
       fname = $3;
       formals = List.rev $5;
       body = List.rev $8 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
    typ ID                    { [($1,$2)]     }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
   expr SEMIC { Expr($1) }
 | LBRACE stmt_list RBRACE                  { Block(List.rev $2)    }
 | RETURN expr_opt SEMIC                    { Return $2             }
 | typ ID SEMIC { Bind($1, $2) }
 | typ ID ASSIGN expr SEMIC { BindAssign($1, $2, $4)}
 | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
 | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)        }
 | WHILE LPAREN expr RPAREN stmt { While ($3, $5) }
 | FOR LPAREN stmt expr SEMIC expr RPAREN stmt { For ($3,$4,$6, $8) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          {$1 }

expr:
   PLIT { PLiteral($1) }
 | STRLITERAL { StrLiteral($1) }
 | LITERAL          { Literal($1) }
 | FLIT { Fliteral($1)}
 | BLIT { BoolLit($1)}
 | RLIT { RLiteral($1) }
 | accessible { $1 }
 | ID LPAREN args_opt RPAREN { Call($1, $3) }
  | accessible ASSIGN expr {Assign ($1, $3) }
  /* math */
  | expr PLUS   expr { Binop($1, Add, $3) }
  | expr MINUS  expr { Binop($1, Sub, $3) }
  | expr TIMES  expr { Binop($1, Mult, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr MOD    expr { Binop($1, Mod, $3) }
  | LPAREN expr RPAREN { $2 }
  /* logic */
  | expr EQ     expr { Binop($1, Equal, $3)   }
  | expr NEQ    expr { Binop($1, Neq,   $3)   }
  | expr LT     expr { Binop($1, Less,  $3)   }
  | expr LEQ    expr { Binop($1, Leq,   $3)   }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3)   }
  | expr AND    expr { Binop($1, And,   $3)   }
  | expr OR     expr { Binop($1, Or,    $3)   }
  /* Unary */
  | MINUS expr %prec NOT { Unop(Neg, $2)      }
  | NOT expr         { Unop(Not, $2)}
  /* note constructor */
  | NEW NOTE LPAREN args_opt RPAREN { CreateNote($4) }
  /* note access */
  | accessible DOT RHYTHM { NoteAccess($1, R) }
  | accessible DOT PITCH  { NoteAccess($1, P) }
  | accessible DOT RHYTHM ASSIGN expr { NoteAssign($1, R, $5) }
  | accessible DOT PITCH ASSIGN expr { NoteAssign($1, P, $5) }
 /* array constructor and accessor */
```

```
  | NEW typ LBRACK expr RBRACK { CreateArray($2, $4) }
  | LBRACK args_opt RBRACK { ArrayLiteral($2) }
  | accessible DOT LENGTH { ArrayLength($1) }

accessible:
    ID { Id($1) }
  | accessible LBRACK expr RBRACK { ArrayAccess($1, $3) }

args_opt:
    /* nothing */ { [] }
  | args_list  { List.rev $1 }

args_list:
    expr                    { [$1] }
  | args_list COMMA expr { $3 :: $1 }

typ:
    VOID  { Void }
  | INT { Int }
  | BOOL { Bool }
  | NOTE { Note }
  | FLOAT { Float }
  | STRING { String }
  | typ LBRACK RBRACK { Array($1)}
```

ast.ml

```
type typ = Note | Void | Int | String | Float | Bool | Rhythm | Pitch |
Array of typ

type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater
| Geq |
          And | Or

type uop = Neg | Not
type prop = R | P

type bind = typ * string

type expr =
    PLiteral of string
  | StrLiteral of string
  | RLiteral of string
  | ArrayLiteral of expr list
  | Call of string * expr list
  | Noexpr
  | Literal of int
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Fliteral of string
  | BoolLit of bool
  | Assign of expr * expr
  | Id of string
  | CreateNote of expr list
  | NoteAccess of expr * prop
  | NoteAssign of expr * prop * expr
  | CreateArray of typ * expr
  | ArrayAccess of expr * expr
  | ArrayLength of expr


type statement =
    Expr of expr
  | Block of statement list
  | Return of expr
  | Bind of bind
  | BindAssign of typ * string * expr
  | If of expr * statement * statement
  | For of statement * expr * expr * statement
  | While of expr * statement

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    body : statement list;
  }

type program = statement list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | String -> "string"
  | Note -> "note"
  | Rhythm -> "rhythm"
  | Pitch -> "pitch"
  | Array t -> string_of_typ t ^ "[]"

let string_of_prop = function
    R -> "rhythm"
  | P -> "pitch"

let rec string_of_expr = function
    PLiteral(l) -> l
    |     Literal(l) -> string_of_int l
    |     StrLiteral(l) ->  "\""^l^"\""
  | RLiteral(l) -> l
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | CreateNote(el) ->
      "new Note(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Id(s) -> s
  | NoteAccess(s, p) -> string_of_expr s ^ "." ^ string_of_prop p
  | NoteAssign(s, p, e) -> string_of_expr s ^ "." ^ string_of_prop p ^ " =
" ^ string_of_expr e
  | ArrayLiteral(l) -> "[" ^ (string_of_list (List.map string_of_expr l)) ^
"]"
  | ArrayAccess(s, e) -> string_of_expr s ^ "[" ^ string_of_expr e ^ "]"
  | ArrayLength(l) -> "length of " ^ string_of_expr l (*TODO: make this
better *)
  | CreateArray(t, e) -> "new " ^ string_of_typ t ^ "[" ^ string_of_expr e
^ "]"

and string_of_list l =
  let add_comma a item =
    a ^ item ^"," in
  List.fold_left add_comma "" l

let rec string_of_stmt = function
    Expr(expr) -> string_of_expr expr ^ ";\n"
  |     Block(stmts) ->
          "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
  | Bind(t, id) -> string_of_typ t ^ " " ^ id ^ ";\n"
  | BindAssign(t, id, e) -> string_of_typ t ^ " " ^ id ^ " = " ^
string_of_expr e ^ ";\n"
  | 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(s1, e2, e3, s) ->
    "for (" ^ string_of_stmt s1 ^ string_of_expr e2 ^ " ; " ^
    string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (stmts, funcs) =
  String.concat "" (List.map string_of_stmt (List.rev stmts)) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for printing it
*)

open Ast

type sexpr = typ * sx
and sx =
        SPLiteral of string
  | SStrLiteral of string
  | SLiteral of int
  | SRLiteral of string
  | SFliteral of string
  | SBoolLit of bool
  | SArrayLiteral of sexpr list
  | SCall of string * sexpr list
  | SNoexpr
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SId of string
  | SCreateNote of sexpr list
  | SNoteAccess of sexpr * prop
  | SNoteAssign of sexpr * prop * sexpr
  | SCreateArray of typ * sexpr
  | SArrayAccess of sexpr * sexpr
  | SArrayLength of sexpr

type sstatement =
    SExpr of sexpr
  | SBlock of sstatement list
  | SReturn of sexpr
  | SBind of bind
  | SBindAssign of typ * string * sexpr
  | SIf of sexpr * sstatement * sstatement
  | SFor of sstatement * sexpr * sexpr * sstatement
  | SWhile of sexpr * sstatement

type sfunc_decl = {
    styp : typ;
    sfname : string;
    sformals : bind list;
    sbody : sstatement list;
  }

type sprogram = sstatement list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SPLiteral(l) -> l
  | SStrLiteral(l) -> "\"" ^ l ^ "\""
  | SLiteral(l) -> string_of_int l
  | SRLiteral(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SCreateNote(el) ->
      "new Note(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAssign(v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
  | SId(s) -> s
  | SNoteAccess(s, p) -> string_of_sexpr s ^ "." ^ string_of_prop p
  | SNoteAssign(s, p, e) -> string_of_sexpr s ^ "." ^ string_of_prop p ^ "
= " ^ string_of_sexpr e
  (*TODO: Make it say something more helpful?*)
  | SArrayLiteral(_) -> "ARRAY LITERAL"
  | SArrayAccess(_, _) -> "ARRAY ACCESSING"
  | SArrayLength(_) -> "ARRAY LENGTH"
  | SCreateArray(_, _) -> "CREATING A NEW ARRAY"
  | SNoexpr -> ""
                      ) ^ ")"

let  rec  string_of_sstmt = function
    SExpr(expr) -> string_of_sexpr expr ^ ";\n"
  | SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n"
  | SBind(t, id) -> string_of_typ t ^ " " ^ id ^ ";\n"
  | SBindAssign(t, id, e) -> string_of_typ t ^ " " ^ id ^ " = " ^
string_of_sexpr e ^ ";\n"
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(s1, e2, e3, s) ->
    "for (" ^ string_of_sstmt s1 ^ string_of_sexpr e2 ^ " ; " ^
    string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt
s


let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (stmts, funcs) =
  String.concat "" (List.map string_of_sstmt stmts) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

semant.ml

```
(* Semantic checking for the E-CATZ compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (statements, functions) =

  (* Verify a list of bindings has no void types or duplicate names *)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
      (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      |     ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
        raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (**** Check functions ****)

  (* Collect function declarations for built-in functions with one
argument: no bodies *)
  let built_in_decls_partial_partial =
    let add_bind map (name, ty) = StringMap.add name {
      typ = Void;
      fname = name;
      formals = [(ty, "x")];
      body = [] } map
    in List.fold_left add_bind StringMap.empty [ ("print", Int); ("prints",
String);
                                                ("printb", Bool);
("printf", Float); ]
  in

  (* Built-ins with 2 arguments *)
  let built_in_decls_partial =
    let add_bind map (name, ty1, ty2, r) = StringMap.add name {
      typ = r;
      fname = name;
      formals = [(ty1, "x"); (ty2, "y")];
      body = [] } map
    in List.fold_left add_bind built_in_decls_partial_partial
      [ ("randInt", Int, Int, Int); ("write", Array Note, String, Void);
         ("writeN", Array (Array Note), String, Void); ]
  in

  (* Built-ins with no arguments *)
  let built_in_decls =
   StringMap.add "seed" {
    typ = Void;
    fname = "seed";
    formals = [];
    body = [] } built_in_decls_partial
    in

  (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.fname ^ " may not be defined"
    and dup_err = "duplicate function " ^ fd.fname
    and make_err er = raise (Failure er)
    and n = fd.fname (* Name of the function *)
    in match fd with (* No duplicate functions or redefinitions of
built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in

  (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

  (* Raise an exception if the given rvalue type cannot be assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if (lvaluet = rvaluet || (lvaluet = Note && rvaluet = Pitch)) then
lvaluet else raise (Failure err)
  in

  (* Return a variable from our local symbol table *)
  let type_of_identifier s symbols =
    try StringMap.find s symbols
    with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in
  (* Return a semantically-checked expression, i.e., with a type *)
  let rec expr symbols = function
      PLiteral l -> (Pitch, SPLiteral l)
    | StrLiteral l -> (String, SStrLiteral l)
    | RLiteral l -> (Rhythm, SRLiteral l)
    | Literal  l -> (Int, SLiteral l)
    | Fliteral l -> (Float, SFliteral l)
    | BoolLit l  -> (Bool, SBoolLit l)
    | Noexpr     -> (Void, SNoexpr)
    | Id s       -> (type_of_identifier s symbols, SId s)
    | Assign(var, e) as ex ->
              let (lt, var') = expr symbols var
              and (rt, e') = expr symbols e in
              let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
                string_of_typ rt ^ " in " ^ string_of_expr ex
              in (check_assign lt rt err, SAssign((lt, var'), (rt, e')))
    | Unop(op, e) as ex ->
          let (t, e') = expr symbols e in
          let ty = match op with
            Neg when t = Int || t = Float -> t
          | Not when t = Bool -> Bool
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
    | Binop(e1, op, e2) as e ->
              let (t1, e1') = expr symbols e1
              and (t2, e2') = expr symbols e2 in
              (* All binary operators require operands of the same type *)
              let same = t1 = t2 in
              (* Determine expression type based on operator and operand
types *)
              let ty = match op with
```

```
                Add | Sub | Mult | Div | Mod when same && t1 = Int   -> Int
              | Add | Sub | Mult | Div when same && t1 = Float -> Float
              | Equal | Neq  when same && (t1 = Int || t1 = Float || t1 =
Pitch || t1 = Rhythm) -> Bool
              | Less | Leq | Greater | Geq
                        when same && (t1 = Int || t1 = Float) -> Bool
              | And | Or when same && t1 = Bool -> Bool
              | _ -> raise (
            Failure ("illegal binary operator " ^
                          string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                          string_of_typ t2 ^ " in " ^ string_of_expr e))
              in (ty, SBinop((t1, e1'), op, (t2, e2')))
     | Call(fname, args) as call ->
         let fd = find_func fname in
         let param_length = List.length fd.formals in
         if List.length args != param_length then
           raise (Failure ("expecting " ^ string_of_int param_length ^
                           " arguments in " ^ string_of_expr call))
         else let check_call (ft, _) e =
           let (et, e') = expr symbols e in
           let err = "illegal argument found " ^ string_of_typ et ^
             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
           in (check_assign ft et err, e')
         in
         let args' = List.map2 check_call fd.formals args
         in (fd.typ, SCall(fname, args'))
    (* Notes must be created with exactly two arguments, pitch and rhythm
*)
     | CreateNote(args) as call ->
         if List.length args != 2 then
           raise (Failure ("expecting " ^ string_of_int 2 ^
                           " arguments in " ^ string_of_expr call))
         else let check_call ft e =
           let (et, e') = expr symbols e in
           let err = "illegal argument found " ^ string_of_typ et ^
             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
           in (check_assign ft et err, e')
         in
         let args' = List.map2 check_call [Pitch; Rhythm] args
         in (Note, SCreateNote(args'))
     | NoteAccess(s, p) ->
         let (typ, s') = expr symbols s in
         (* Make sure that dot access is being used on a note *)
```

```
        if (typ=Note) then
          let rtyp = match p with
              P -> Pitch
            | R -> Rhythm
          in
          (rtyp, SNoteAccess((typ, s'), p))
        else
          raise (Failure ("Accessing pitch or rhythm may only be used on
identifier of type Note. " ^ string_of_expr s ^ " is not a Note." ))
    | NoteAssign(s, p, e) as ex ->
        let (typ, s') = expr symbols s in
        (* Make sure that dot access is being used on a note *)
        if (typ=Note) then
          let lt = match p with
              P -> Pitch
            | R -> Rhythm
          in
          let (rt, e') = expr symbols e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SNoteAssign((typ, s'), p, (rt, e')))
        else
          raise (Failure ("Attempting to assign a Note to an identifier of
the wrong type" ))
    | ArrayLength(s) ->
       let (typ, s') = expr symbols s in
       (* Check that we're trying to get the length of an array*)
       let _ = match typ with
          Array t -> t
        | _ ->  raise (Failure ("Only arrays have lengths you fool. " ^
string_of_expr s ^ " is not an array." ))
       in (Int, SArrayLength((typ, s')))
    | ArrayLiteral(l) ->
        let param_length = List.length l in
        (* if someone types in [], it raises an error *)
        if (param_length = 0) then
          raise (Failure ("Array literals must have at least one
argument"))
        else
          (* Assume the first type of the array is the right type *)
          let (first_type, _) = expr symbols (List.hd l) in
          (* If a pitch literal appears in an array literal, treat it as a
note *)
```

```
          let true_type = match first_type with
              Pitch -> Note
            | _ -> first_type
          in
          (* check that each element in an array literal matches the first
type given *)
          let check_call e =
          let (et, e') = expr symbols e in
          let err = "argument of incorrect type included: " ^ string_of_typ
first_type ^
            " expected, but " ^ string_of_typ et ^ " found"
          in (check_assign true_type et err, e')
        in
        let args' = List.map check_call l
        in ((Array true_type), SArrayLiteral(args'))
    | CreateArray(t, e) ->
        let (et, e') = expr symbols e in
        let et' =
          (match et with
              Int -> et
            | _ -> raise(Failure("Arrays can only be initialized with an
integer length. " ^
                    string_of_expr e ^ " is not an int."))
          )
        in ((Array t), SCreateArray(t, (et', e')))
    | ArrayAccess(s, e) ->
        (* check that brackets are being used with arrays *)
        let (typ, s') = expr symbols s in
        let array_typ = match typ with
            Array t -> t
          | _ ->  raise (Failure ("Only arrays can be accessed with bracket
notation. " ^ string_of_expr s ^ " is not an array." ))
        in
        (* check that it is an int inside the bracket access *)
        let (et, e') = expr symbols e in
        let et' = match et with
            Int -> et
          | _ -> raise(Failure("Array indexing must be done with integers.
" ^ string_of_expr e ^ " is not an int."))
        in (array_typ, SArrayAccess((typ, s'), (et', e')))

  in
  let check_bool_expr s e =
    let (t', e') = expr s e
    and err = "expected Boolean expression in " ^ string_of_expr e
    in if t' != Bool then raise (Failure err) else (t', e')
  in

  (* Return a semantically-checked statement i.e. containing sexprs *)
  (* move this function out of the regular check_stmt so that we can
maintain our symbol table easier
      especially when working with returns *)
  let rec check_stmt_list symbols typ = function
      [Return _ as s] -> ([fst(check_stmt symbols typ s)], symbols)
    | Return _ :: _   -> raise (Failure "nothing may follow a return")
    | Block sl :: ss  -> check_stmt_list symbols typ (sl @ ss) (* Flatten
blocks *)
    | s :: ss         ->
        let (single, new_symbol) = check_stmt symbols typ s in
        let (rest, final_symbols) = check_stmt_list new_symbol typ ss in
        (single :: rest, final_symbols)
    | []              -> ([], symbols)

  and check_stmt symbols typ = function
      Expr e -> (SExpr (expr symbols e), symbols)
    | Return e -> let (t, e') = expr symbols e in
      if t = typ then (SReturn (t, e'), symbols)
      else raise (
  Failure ("return gives " ^ string_of_typ t ^ " expected " ^
     string_of_typ typ ^ " in " ^ string_of_expr e))
    (* A block is correct if each statement is correct and nothing
       follows any Return statement.  Nested blocks are flattened. *)
    | Block sl -> (SBlock(fst (check_stmt_list symbols typ sl)), symbols)
    (* make sure that a variable is neither void nor being rebound *)
    | Bind(lt, id) ->
        if (lt = Void) then
          let err ="cannot bind " ^ id ^ " to Void type"
          in raise (Failure err)
        else
        if (StringMap.mem id symbols) then
          let err = "cannot bind " ^ id ^ " as it already is bound"
          in raise (Failure err)
        else
        let sym' = StringMap.add id lt symbols in ( SBind(lt, id), sym' )
    (* same as above but also checks assignment matches bind *)
    | BindAssign(lt, id, e) as ex ->
        if (lt = Void) then
          let err ="cannot bind " ^ id ^ " to Void type"
          in raise (Failure err)
        else
        if (StringMap.mem id symbols) then
          let err = "cannot bind " ^ id ^ " as it already is bound"
          in raise (Failure err)
        else
          let (rt, e') = expr symbols e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_stmt ex
          in let typ = check_assign lt rt err
          in let sym' = StringMap.add id lt symbols
          in ( SBindAssign(typ, id, (rt, e')), sym' )
    | If(p, b1, b2) -> (SIf(check_bool_expr symbols p, fst(check_stmt
symbols typ b1), fst(check_stmt symbols typ b2)), symbols)
    | For(s1, e2, e3, s) ->
      let (sstmt, symbols') = check_stmt symbols typ s1 in
      (SFor( sstmt, check_bool_expr symbols' e2, expr symbols' e3,
fst(check_stmt symbols' typ s)), symbols)
    | While(e, s) -> (SWhile(check_bool_expr symbols e, fst(check_stmt
symbols typ s)), symbols)

  in

  let check_function func =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "formal" func.formals;

    (* Build local symbol table of variables for this function *)
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty
m)
                    StringMap.empty (func.formals)
    in
    let (sl, _) = check_stmt symbols func.typ (Block func.body) in

(* body of check_function *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      sbody = match sl with
          SBlock(sl) -> sl
        | _ -> raise (Failure ("internal error: block didn't become a
block?"))
    }

  in (fst (check_stmt_list StringMap.empty Void (List.rev statements)),
List.map check_function functions)
```

codegen.ml

```
module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

let translate (statements, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
let the_module = L.create_module context "E-CATZ" in

  (* Get types from the context *)
let i32_t      = L.i32_type    context
and i8_t   = L.i8_type     context
and void_t = L.void_type   context
and i1_t       = L.i1_type     context
and float_t    = L.double_type context
and string_t   = L.pointer_type (L.i8_type context)
in
let pitch_t    = string_t
and rhythm_t   = string_t
in
let note_t     = L.struct_type context [| pitch_t; rhythm_t |]
    in
let rec ltype_of_typ = function
      A.String    -> string_t
    | A.Note -> note_t
    | A.Void -> void_t
```

```
     | A.Int  -> i32_t
     | A.Bool  -> i1_t
     | A.Float -> float_t
     | A.Rhythm -> rhythm_t
     | A.Pitch -> pitch_t
     | A.Array t ->  array_t t
and array_t t = L.struct_type context [| L.pointer_type (ltype_of_typ t);
i32_t |]
    in

(* all of our beautiful built-in c functions *)
  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
      L.declare_function "printf" printf_t the_module in

let write_t: L.lltype =
      L.var_arg_function_type i32_t [| array_t A.Note ; string_t |] in
  let write_func : L.llvalue =
      L.declare_function "write" write_t the_module in

let writeN_t: L.lltype =
      L.var_arg_function_type i32_t [| array_t (A.Array A.Note) ; string_t
|] in
  let writeN_func : L.llvalue =
      L.declare_function "writeN" writeN_t the_module in

let randInt_t: L.lltype =
      L.var_arg_function_type i32_t [| i32_t; i32_t |] in
  let randInt_func : L.llvalue =
      L.declare_function "randInt" randInt_t the_module in

let seed_t: L.lltype =
      L.var_arg_function_type i32_t [| |] in
  let seed_func : L.llvalue =
      L.declare_function "seed" seed_t the_module in

(* put all code that isn't inside a function inside a big dummy main
function *)
    let build_main stmts funcs=
        let main : L.lltype =
            L.var_arg_function_type i32_t [| |] in
        let main_func : L.llvalue =
            L.define_function "main" main the_module in
        let builder = L.builder_at_end context (L.entry_block main_func) in


  (* Format to String *)
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder
  and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in

 (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
     let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
      let function_decl m fdecl =
        let name = fdecl.sfname
        and formal_types =
    Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
        in let ftype = L.function_type (ltype_of_typ fdecl.styp)
formal_types in
        StringMap.add name (L.define_function name ftype the_module, fdecl)
m in
      List.fold_left function_decl StringMap.empty functions in

      (* because of how data is stored and loaded into arrays
        this function allows you to get data in ids and in array brackets
(x[3])
        without directly loading it. When this function is called
        another piece of the code knows it's responsible for loading it
later.
        We were having issues with double loads *)
      let rec access builder locals ((t, e) : sexpr) =
      match e with
          SId s       -> StringMap.find s locals
        | SArrayAccess(s, e) ->
            let array_struct = access builder locals s in
            let array_ptr = L.build_struct_gep array_struct 0 "array_ptr"
builder in
            let array = L.build_load array_ptr "array" builder in
            let index = expr builder locals e in
            let element_ptr = L.build_gep array [| index |] "array_access"
builder in
            element_ptr
        | _ -> raise(Failure("This expression " ^ string_of_sexpr (t, e) ^
" does not access data."))
```

```
      (* This function was created for the same reason as above except for
when dealing with Note
        structs instead. Again a double load problem. Also, it makes so
that pitch literals actually
        act as pitches instead of being a note constructor. This is only
used when doing dot access to a
        note. Additionally, if it doesn't get the data it wants,
        it just makes a C4 note instead of throwing an error *)
      and get_needed_struct_value builder locals ((_, e) : sexpr) =
      match e with
          SPLiteral p -> L.build_global_stringptr p "pitch" builder
        | SNoteAccess(s, A.P) ->
            let note_struct = access builder locals s in
            let element_ptr = L.build_struct_gep note_struct 0
"element_ptr" builder in
            L.build_load element_ptr "element" builder
        | SNoteAccess(s, A.R) ->
            let note_struct = access builder locals s in
            let element_ptr = L.build_struct_gep note_struct 1
"element_ptr" builder in
            L.build_load element_ptr "element" builder
        | SRLiteral r -> L.build_global_stringptr r "rhythm" builder
        | SNoteAssign(s, p, e) ->
            let note_struct = access builder locals s in
            let idx = (match p with
                A.P -> 0
              | A.R -> 1
            ) in
            let element_ptr = L.build_struct_gep note_struct idx
"element_ptr" builder in
            let value = get_needed_struct_value builder locals e in
            ignore(L.build_store value element_ptr builder); value
        | _ -> L.build_global_stringptr "C4" "pitch" builder


      (* Construct code for an expression; return its value *)
      and expr builder locals ((_, e) : sexpr) =
      match e with
          SPLiteral p ->
            (* a pitch literal on its own acts as a note constructor*)
            let note_ptr = L.build_malloc note_t "note_ptr" builder in
            let pitch_ptr = L.build_struct_gep note_ptr 0 "pitch_ptr"
```

```
builder in
            let pitch = L.build_global_stringptr p "pitch" builder in
            ignore(L.build_store pitch pitch_ptr builder);
            let rhythm_ptr = L.build_struct_gep note_ptr 1 "rhythm_ptr"
builder in
            (* default quarter note rhythm *)
            let rhythm = expr builder locals ((A.Rhythm, SRLiteral("qr")))
in
            ignore(L.build_store rhythm rhythm_ptr builder);
            L.build_load note_ptr "note" builder
        | SLiteral i  -> L.const_int i32_t i
        | SRLiteral r -> L.build_global_stringptr r "rhythm" builder
        | SStrLiteral a -> L.build_global_stringptr a "str" builder
        | SNoexpr     -> L.const_int i32_t 0
        | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
        | SFliteral l -> L.const_float_of_string float_t l
        | SId s       -> L.build_load (StringMap.find s locals) s builder
      | SAssign (s, e) -> let e' = expr builder locals e in
                          ignore(L.build_store e' (access builder locals s)
builder); e'
      | SCall ("write", [e1; e2]) ->
      L.build_call write_func [| (expr builder locals e1); (expr builder
locals e2) |]
        "write" builder
      | SCall ("writeN", [e1; e2]) ->
            L.build_call writeN_func [| (expr builder locals e1); (expr
builder locals e2) |]
              "writeN" builder
      | SCall ("print", [e]) | SCall ("printb", [e])  ->
            L.build_call printf_func [| int_format_str; (expr builder
locals e ) |]
            "printf" builder
      | SCall ("prints", [e]) ->
            L.build_call printf_func [| str_format_str ; (expr builder
locals e) |]
            "printf" builder
      | SCall ("printf", [e]) ->
            L.build_call printf_func [| float_format_str ; (expr builder
locals e) |]
              "printf" builder
      | SCall ("randInt", [e1; e2])  ->
          L.build_call randInt_func [| (expr builder locals e1 );  (expr
builder locals e2 ) |]
```

```
                "randInt" builder
       | SCall ("seed", [])  ->
           L.build_call seed_func [| |]
                 "seed" builder
       | SCall (f, args) ->
          let (fdef, fdecl) = StringMap.find f function_decls in
          let llargs = List.rev (List.map (expr builder locals) (List.rev
args)) in
          let result = (match fdecl.styp with
                              A.Void -> ""
                            | _ -> f ^ "_result") in
                L.build_call fdef (Array.of_list llargs) result builder
       | SCreateNote(args) ->
           let note_ptr = L.build_malloc note_t "note_ptr" builder in
           let pitch_ptr = L.build_struct_gep note_ptr 0 "pitch_ptr" builder
in
           let pitch = get_needed_struct_value builder locals (List.hd args)
in
           ignore(L.build_store pitch pitch_ptr builder);
           let rhythm_ptr = L.build_struct_gep note_ptr 1 "rhythm_ptr"
builder in
           let rhythm = expr builder locals (List.nth args 1) in
           ignore(L.build_store rhythm rhythm_ptr builder);
           L.build_load note_ptr "note" builder
       | SUnop(op, ((t, _) as e)) ->
           let e' = expr builder locals e in
         (match op with
           A.Neg when t = A.Float -> L.build_fneg
         | A.Neg                  -> L.build_neg
         | A.Not                  -> L.build_not) e' "tmp" builder
       | SNoteAccess(s, p) ->
           let note_struct = access builder locals s in
           let idx = (match p with
               A.P -> 0
             | A.R -> 1
           ) in
           let element_ptr = L.build_struct_gep note_struct idx
"element_ptr" builder in
           L.build_load element_ptr "element" builder
       | SNoteAssign(s, p, e) ->
           let note_struct = access builder locals s in
           let idx = (match p with
               A.P -> 0
```

```
            | A.R -> 1
          ) in
          let element_ptr = L.build_struct_gep note_struct idx
"element_ptr" builder in
          let value = get_needed_struct_value builder locals e in
          ignore(L.build_store value element_ptr builder); value
      | SCreateArray(t, l) ->
          (* create an empty array of desired length *)
          let l' = expr builder locals l in
          let t' = ltype_of_typ t in
          let struct_ptr = L.build_malloc (array_t t) "struct_ptr" builder
in
          let array_in_struct_ptr = L.build_struct_gep struct_ptr 0
"array_in_struct_ptr" builder in
          let length_in_struct_ptr = L.build_struct_gep struct_ptr 1
"length_in_struct_ptr" builder in
          ignore(L.build_store l' length_in_struct_ptr builder);
          let array_ptr = L.build_array_malloc t' l' "array_lit" builder
          in
          ignore(L.build_store array_ptr array_in_struct_ptr builder);
          L.build_load struct_ptr "struct" builder
      | SArrayAccess(s, e) ->
          let array_struct = access builder locals s in
          let array_ptr = L.build_struct_gep array_struct 0 "array_ptr"
builder in
          let array = L.build_load array_ptr "array" builder in
          let index = expr builder locals e in
          let element_ptr = L.build_gep array [| index |] "array_access"
builder in
          L.build_load element_ptr "element" builder
      | SArrayLength(s) ->
         let array_struct = access builder locals s in
         let length_ptr = L.build_struct_gep array_struct 1 "length_ptr"
builder in
         L.build_load length_ptr "length" builder
      | SArrayLiteral(e) ->
          let length = L.const_int i32_t (List.length e) in
          let t = fst (List.hd e) in
          let t' = ltype_of_typ t in
          let struct_ptr = L.build_malloc (array_t t) "struct_ptr" builder
in
          let array_in_struct_ptr = L.build_struct_gep struct_ptr 0
"array_in_struct_ptr" builder in
```

```
          let length_in_struct_ptr = L.build_struct_gep struct_ptr 1
"length_in_struct_ptr" builder in
          ignore(L.build_store length length_in_struct_ptr builder);
          let array_ptr = L.build_array_malloc t' length "array_lit"
builder
          in
          (* iterate through each element in the array literal and build it
*)
          let build_element idx e1 =
            let e1' = expr builder locals e1 in
            let llindex = L.const_int i32_t idx in
            let element_ptr = L.build_gep array_ptr [| llindex |]
"build_elem" builder in
            let casted_element_ptr = L.build_pointercast element_ptr
(L.pointer_type t') "casted" builder in
            ignore(L.build_store e1' casted_element_ptr builder);
          in ignore(List.iteri build_element e);
          ignore(L.build_store array_ptr array_in_struct_ptr builder);
L.build_load struct_ptr "struct" builder
      | SBinop ((A.Float,_ ) as e1, op, e2) ->
          let e1' = expr builder locals e1
          and e2' = expr builder locals e2  in
          (match op with
            A.Add     -> L.build_fadd
          | A.Sub     -> L.build_fsub
          | A.Mult    -> L.build_fmul
          | A.Div     -> L.build_fdiv
          | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
          | A.Neq     -> L.build_fcmp L.Fcmp.One
          | A.Less    -> L.build_fcmp L.Fcmp.Olt
          | A.Leq     -> L.build_fcmp L.Fcmp.Ole
          | A.Greater -> L.build_fcmp L.Fcmp.Ogt
          | A.Geq     -> L.build_fcmp L.Fcmp.Oge
          | A.And | A.Or | A.Mod ->
              raise (Failure "internal error: semant should have rejected
and/or/mod on float")
          ) e1' e2' "tmp" builder
            | SBinop (e1, op, e2) ->
          let e1' = expr builder locals e1
          and e2' = expr builder locals e2  in
          (match op with
            A.Add     -> L.build_add
          | A.Sub     -> L.build_sub
          | A.Mult    -> L.build_mul
          | A.Div     -> L.build_sdiv
          | A.Mod     -> L.build_srem
          | A.And     -> L.build_and
          | A.Or      -> L.build_or
          | A.Equal   -> L.build_icmp L.Icmp.Eq
          | A.Neq     -> L.build_icmp L.Icmp.Ne
          | A.Less    -> L.build_icmp L.Icmp.Slt
          | A.Leq     -> L.build_icmp L.Icmp.Sle
          | A.Greater -> L.build_icmp L.Icmp.Sgt
          | A.Geq     -> L.build_icmp L.Icmp.Sge
          ) e1' e2' "tmp" builder

      in

      (* LLVM insists each basic block end with exactly one "terminator"
         instruction that transfers control.  This function runs "instr
builder"
         if the current block does not already have a terminator.  Used,
         e.g., to handle the "fall off the end of the function" case. *)
      let add_terminal builder instr =
        match L.block_terminator (L.insertion_block builder) with
    Some _ -> ()
        | None -> ignore (instr builder) in

    let add_local m (t, n) p b =
          L.set_value_name n p;
    let local = L.build_alloca (ltype_of_typ t) n b in
          ignore (L.build_store p local b);
    StringMap.add n local m

    in

    (* this function is used to make it so we can maintain our master
locals map
      of variables *)
    let rec build_stmt_list ftyp func builder locals = function
          hd :: tl -> let (b, l) = build_stmt ftyp func builder locals hd
in
            build_stmt_list ftyp func b l tl
        | [] -> (builder, locals)

    and build_stmt ftyp func builder locals = function
```

```
          SBlock sl -> build_stmt_list ftyp func builder locals sl
        | SExpr e -> ignore(expr builder locals e ); (builder, locals)
        | SReturn e -> ignore(match ftyp with
                                A.Void -> L.build_ret_void builder
                              | _ -> L.build_ret (expr builder locals e )
builder );
                               (builder, locals)
         | SIf (predicate, then_stmt, else_stmt) ->
         let bool_val = expr builder locals predicate in
       let merge_bb = L.append_block context "merge" func in
         let build_br_merge = L.build_br merge_bb in (* partial function *)

       let then_bb = L.append_block context "then" func in
       add_terminal ( fst (build_stmt ftyp func (L.builder_at_end context
then_bb) locals then_stmt))
         build_br_merge;

       let else_bb = L.append_block context "else" func in
       add_terminal (fst (build_stmt ftyp func (L.builder_at_end context
else_bb) locals else_stmt))
         build_br_merge;

       ignore(L.build_cond_br bool_val then_bb else_bb builder);
       (L.builder_at_end context merge_bb, locals)
        | SBindAssign (ty, name, sexpr) -> let x = expr builder locals
sexpr in
                                          let locals' = add_local locals
(ty, name) x builder in
                                          (builder, locals')
        | SBind (ty, name) -> let locals' = add_local locals (ty, name)
(L.const_int i32_t 0) builder in (builder, locals')
        | SWhile (predicate, body) ->
          let pred_bb = L.append_block context "while" func in
          ignore(L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body" func in
          add_terminal (fst (build_stmt ftyp func (L.builder_at_end context
body_bb) locals body)) (* (stmt (L.builder_at_end context body_bb) body) *)
            (L.build_br pred_bb);

          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr pred_builder locals predicate in
          let merge_bb = L.append_block context "merge" func in
          ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
          (L.builder_at_end context merge_bb, locals)

            (* Implement for loops as while loops *)
          | SFor (s1, e2, e3, body) -> let (b, l) = build_stmt ftyp func
builder locals s1 in
          build_stmt ftyp func b l
            (  SWhile (e2, SBlock [body ; SExpr e3]) )
    in

        (* this is the code that actually starts to build statements that
are not inside of functions
          it starts with an empty locals table that is populated as it
reads Binds in check_stmt*)
        let (builder, _) =
            build_stmt A.Int main_func builder StringMap.empty (SBlock
stmts);
        in
        add_terminal builder (L.build_ret (L.const_int i32_t 0));

        (* Fill in the body of the given function *)
        let build_function_body fdecl =
          let (the_function, _) = StringMap.find fdecl.sfname
function_decls in
          let builder = L.builder_at_end context (L.entry_block
the_function) in

       (* Construct the function's "locals": formal arguments and locally
             declared variables.  Allocate each on the stack, initialize
their
             value, if appropriate, and remember their values in the
"locals" map *)
          let local_vars =
            let add_formal m (t, n) p =
              L.set_value_name n p;
        let local = L.build_alloca (ltype_of_typ t) n builder in
              ignore (L.build_store p local builder);
        StringMap.add n local m
            in

          List.fold_left2 add_formal StringMap.empty fdecl.sformals
              (Array.to_list (L.params the_function)) in
```

```
         (* Build the code for each statement in the function *)
          let (builder, _) = build_stmt fdecl.styp the_function builder
local_vars (SBlock fdecl.sbody)  in

          (* Add a return if the last block falls off the end *)
          add_terminal builder (match fdecl.styp with
              A.Void -> L.build_ret_void
            | A.Float -> L.build_ret (L.const_float float_t 0.0)
            | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
        in
        List.iter build_function_body funcs
    in

    build_main statements functions;
    the_module
```

ecatz.ml

```
(* Top-level of the E-CATZ compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./ecatz.native [-a|-s|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;
```

```
  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.tokenize lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate
sast))
    | Compile -> let m = Codegen.translate sast in
      Llvm_analysis.assert_valid_module m;
      print_string (Llvm.string_of_llmodule m)
```

stdlib.catz

```
def int noteToInt(Note n) {
  Note[] standard = [ C0, C#0, D0, D#0, E0, F0, F#0, G0, G#0, A0, A#0, B0,
                      C1, C#1, D1, D#1, E1, F1, F#1, G1, G#1, A1, A#1, B1,
                      C2, C#2, D2, D#2, E2, F2, F#2, G2, G#2, A2, A#2, B2,
                      C3, C#3, D3, D#3, E3, F3, F#3, G3, G#3, A3, A#3, B3,
                      C4, C#4, D4, D#4, E4, F4, F#4, G4, G#4, A4, A#4, B4,
                      C5, C#5, D5, D#5, E5, F5, F#5, G5, G#5, A5, A#5, B5,
                      C6, C#6, D6, D#6, E6, F6, F#6, G6, G#6, A6, A#6, B6,
                      C7, C#7, D7, D#7, E7, F7, F#7, G7, G#7, A7, A#7, B7,
                      C8, C#8, D8, D#8, E8, F8, F#8, G8, G#8, A8, A#8, B8,
                      C9, C#9, D9, D#9, E9, F9, F#9, G9, G#9, A9, A#9, B9
];

  Note[] funky =    [ B#0, D@0, D0, E@0, F@0, E#0, G@0, G0, A@0, A0, B@0,
C@0,
                      B#1, D@1, D1, E@1, F@1, E#1, G@1, G1, A@1, A1, B@1,
C@1,
                      B#2, D@2, D2, E@2, F@2, E#2, G@2, G2, A@2, A2, B@2,
C@2,
                      B#3, D@3, D3, E@3, F@3, E#3, G@3, G3, A@3, A3, B@3,
C@3,
                      B#4, D@4, D4, E@4, F@4, E#4, G@4, G4, A@4, A4, B@4,
C@4,
                      B#5, D@5, D5, E@5, F@5, E#5, G@5, G5, A@5, A5, B@5,
C@5,
                      B#6, D@6, D6, E@6, F@6, E#6, G@6, G6, A@6, A6, B@6,
C@6,
                      B#7, D@7, D7, E@7, F@7, E#7, G@7, G7, A@7, A7, B@7,
C@7,
                      B#8, D@8, D8, E@8, F@8, E#8, G@8, G8, A@8, A8, B@8,
C@8,
                      B#9, D@9, D9, E@9, F@9, E#9, G@9, G9, A@9, A9, B@9,
C@9 ];

  for (int i = 0; i < standard.length; i = i + 1) {
    if (standard[i].pitch == n.pitch) {
      return i;
    }
  }
  for (int j = 0; j < standard.length; j = j + 1) {
      if (funky[j].pitch == n.pitch) {
        return j;
      }
    }
  return 0;
}

def Note intToNote(int i) {
  Note[] standard = [ C0, C#0, D0, D#0, E0, F0, F#0, G0, G#0, A0, A#0, B0,
                      C1, C#1, D1, D#1, E1, F1, F#1, G1, G#1, A1, A#1, B1,
                      C2, C#2, D2, D#2, E2, F2, F#2, G2, G#2, A2, A#2, B2,
                      C3, C#3, D3, D#3, E3, F3, F#3, G3, G#3, A3, A#3, B3,
                      C4, C#4, D4, D#4, E4, F4, F#4, G4, G#4, A4, A#4, B4,
                      C5, C#5, D5, D#5, E5, F5, F#5, G5, G#5, A5, A#5, B5,
                      C6, C#6, D6, D#6, E6, F6, F#6, G6, G#6, A6, A#6, B6,
                      C7, C#7, D7, D#7, E7, F7, F#7, G7, G#7, A7, A#7, B7,
                      C8, C#8, D8, D#8, E8, F8, F#8, G8, G#8, A8, A#8, B8,
                      C9, C#9, D9, D#9, E9, F9, F#9, G9, G#9, A9, A#9, B9
];

  return standard[i];

}

def Note addToNote(Note n, int i) {
  int newPitch = noteToInt(n)+i;
  if (newPitch < 0 || newPitch > 119) {
    return n;
  }
  Note output = intToNote(newPitch);
  output.rhythm = n.rhythm;
  return output;
}

def Note octaveChange(Note n, int i) {
  int newPitch = noteToInt(n)+i*12;
  if (newPitch < 0 || newPitch > 119) {
   return n;
  }
  Note output = intToNote(newPitch);
  output.rhythm = n.rhythm;
  return output;
}

def Note[] concatArray(Note[] arr1, Note[] arr2) {
  Note[] output = new Note[arr1.length+arr2.length];
  for (int i = 0; i < arr1.length; i = i + 1) {
    output[i] = arr1[i];
  }
  for (int j = 0; j < arr2.length; j = j + 1) {
    output[j+arr1.length] = arr2[j];
  }
  return output;
}

def Note[][] concatNestedNoteArray(Note[][] arr1, Note[][] arr2) {
  Note[][] output = new Note[][arr1.length+arr2.length];
  for (int i = 0; i < arr1.length; i = i + 1) {
    output[i] = arr1[i];
  }
  for (int j = 0; j < arr2.length; j = j + 1) {
    output[j+arr1.length] = arr2[j];
  }
  return output;
}
```

utils.c

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <string.h>

void writeOctave(int octave, FILE *file)
{
 if (octave < 3)
 {
   for (int i = 0; i < (3-octave); ++i)
   {
     fputs(",", file);
   }
 }
 else
 {
   for (int i = 0; i < (octave-3); ++i)
   {
     fputs("'", file);
   }
 }
}

void writeRhythm(const char* rhythm, FILE *file)
{
 if (strcmp(rhythm, "ts") == 0) { fputs("32", file); }
 else if (strcmp(rhythm, "st") == 0) { fputs("16", file); }
 else if (strcmp(rhythm, "et") == 0) { fputs("8", file); }
 else if (strcmp(rhythm, "qr") == 0) { fputs("4", file); }
 else if (strcmp(rhythm, "hf") == 0) { fputs("2", file); }
 else if (strcmp(rhythm, "wh") == 0) { fputs("1", file); }
}

struct Note{
 char* pitch;
 char* rhythm;
};
struct Harmony{
   struct Note *arr;
   int length;
};
struct Melody{
 struct Harmony *arr;
 int length;
};

void writehelper(const char *pitch, const char* rhythm, FILE *file){
   int octave;
   fprintf(file, "%c", tolower(pitch[0]));

   if (pitch[1] == '#')
   {
     fputs("is", file);
     octave = pitch[2] - '0';
   }
   else if (pitch[1] == '@')
   {
     fputs("es", file);
     octave = pitch[2] - '0';
   }
   else
   {
     octave = pitch[1] - '0';
   }
   writeOctave(octave, file);
   writeRhythm(rhythm, file);
   fputs(" ", file);
}

void write(const struct Harmony m, const char *outputFile)
{
 FILE *file;

 char fullName[1024];
 strcpy(fullName, outputFile);
 strcat(fullName, ".ly");

 file = fopen(fullName, "w+");
 fputs("\\version \"2.22.0\"\n\\score {\n{ ", file);
 for (int i= 0; i<m.length; i++){
   writehelper(m.arr[i].pitch, m.arr[i].rhythm, file);
 }
 fputs(" }\n\\midi { \\tempo 4 = 120 }\n}\n", file);
 fclose(file);
```

```c
}

void writeN(const struct Melody m, const char *outputFile)
{
 FILE *file;

 char fullName[1024];
 strcpy(fullName, outputFile);
 strcat(fullName, ".ly");

 file = fopen(fullName, "w+");
 fputs("\\version \"2.22.0\"\n\\score {\n{ ", file);
 for (int i= 0; i<m.length; i++){
   if(m.arr[i].length==1){
     if (i==m.length-1){
       fputs("\n", file);
     }
     writehelper(m.arr[i].arr[0].pitch, m.arr[i].arr[0].rhythm, file);
   }
   else{
     fputs(" <", file);
     for(int j=0; j<m.arr[i].length;j++){
       writehelper(m.arr[i].arr[j].pitch, "skip:)", file);
     }
     fputs(">", file);
     writeRhythm(m.arr[i].arr[0].rhythm, file);
   }
 }
 fputs(" }\n\\midi { \\tempo 4 = 120 }\n}\n", file);
 fclose(file);
}

void seed() {
 srand ( time(NULL) );
}

int randInt(int min, int max)
{
 return (rand() % (max - min + 1)) + min;
}
```

Appendix B:

testall.sh

```
#!/bin/sh

# Regression testing script for E-CATZ
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the E-CATZ compiler.  Usually "./ecatz.native"
# Try "_build/ecarz.native" if ocamlbuild was unable to create a symbolic link.
ECATZ="./ecatz.native"
#ECATZ="_build/ecatz.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.catz files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
    SignalError "$1 failed on $*"
    return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
```

```
  RunFail() {
      echo $* 1>&2
      eval $* && {
      SignalError "failed: $* did not report an error"
      return 1
      }
      return 0
  }

  Check() {
      error=0
      basename=`echo $1 | sed 's/.*\\///
                           s/.catz//'`
      reffile=`echo $1 | sed 's/.catz$//'`
      basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

      echo -n "$basename..."

      echo 1>&2
      echo "###### Testing $basename" 1>&2

      generatedfiles=""

      generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out temp.catz temp1.catz" &&
      Run "./preprocessor" "$1" ">" "temp.catz" &&
      Run "$ECATZ" "temp.catz" ">" "${basename}.ll" &&
      Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
      Run "$CC" "-o" "${basename}.exe" "${basename}.s" "utils.o" &&
      Run "./${basename}.exe" > "${basename}.out" &&
      Compare ${basename}.out ${reffile}.out ${basename}.diff

      # Report the status and clean up the generated files

      if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
          rm -f $generatedfiles
```

```
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
        else
        echo "###### FAILED" 1>&2
        globalerror=$error
        fi
    }

    CheckWrite() {
        error=0
        basename=`echo $1 | sed 's/.*\\///
                                s/.catz//'`
        reffile=`echo $1 | sed 's/.catz$//'`
        basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

        echo -n "$basename..."

        echo 1>&2
        echo "###### Testing $basename" 1>&2

        generatedfiles=""

        generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.ly temp.catz temp1.catz" &&
        Run "./preprocessor" "$1" ">" "temp.catz" &&
        Run "$ECATZ" "temp.catz" ">" "${basename}.ll" &&
        Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
        Run "$CC" "-o" "${basename}.exe" "${basename}.s" "utils.o" &&
        Run "./${basename}.exe"
        Compare ${basename}.ly ${reffile}.ly ${basename}.diff

        # Report the status and clean up the generated files

        if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
        else
        echo "###### FAILED" 1>&2
        globalerror=$error
        fi
    }

    CheckFail() {
        error=0
        basename=`echo $1 | sed 's/.*\\///
                                s/.catz//'`
        reffile=`echo $1 | sed 's/.catz$//'`
        basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

        echo -n "$basename..."

        echo 1>&2
        echo "###### Testing $basename" 1>&2

        generatedfiles=""

        generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
        RunFail "$ECATZ" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
        Compare ${basename}.err ${reffile}.err ${basename}.diff

        # Report the status and clean up the generated files

        if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
        else
        echo "###### FAILED" 1>&2
        globalerror=$error
        fi
    }

    while getopts kdpsh c; do
        case $c in
        k) # Keep intermediate files
            keep=1
            ;;
        h) # Help
            Usage
            ;;
        esac
    done

    shift `expr $OPTIND - 1`

    LLIFail() {
    echo "Could not find the LLVM interpreter \"$LLI\"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
    }

    which "$LLI" >> $globallog || LLIFail

    if [ ! -f utils.o ]
    then
        echo "Could not find utils.o"
        echo "Try \"make utils.o\""
        exit 1
    fi

    if [ $# -ge 1 ]
    then
        files=$@
    else
        files="tests/test-*.catz tests/fail-*.catz tests/write-*.catz"
```

```
fi

for file in $files
do
    case $file in
    *test-*)
        Check $file 2>> $globallog
        ;;
    *fail-*)
        CheckFail $file 2>> $globallog
        ;;
    *write-*)
        CheckWrite $file 2>> $globallog
        ;;
    *)
        echo "unknown file type $file"
        globalerror=1
        ;;
    esac
done

exit $globalerror
```

Success Tests

test-helloWorld.catz

```
print(1);
```

test-math.catz

```
print(4 + 4);
print(6 / 2);
print(5 * 4);
print(6 - 3);
print(3 % 2);
printf(4.2 + 4.2);
printf(3.1 - 1.2);
printf(4.1 * 5.);
printf(9.9 / 3.);
```

test-precedence.catz

```
int x = 3 * (5+6);
print(x);
x = (3 * 5) + 6;
print(x);
```

test-bind.catz

```
int x;
x = 3;
print(x);
```

test-bindAssign.catz

```
int x = 3;
print(x);
```

test-sequentialAssign.catz

```
int x = 5;
int y = x = 3;
print(y);
print(x);
```

test-comment.catz

```
/* ignore me */

print(3);

/* ignore me too

and me

and also me

*/
```

test -String.catz

```
prints("abc");
```

test-neg.catz

```
int i= -1;
if (i<0){
    print (i);
}
```

test-not.catz

```
if (!false){
    print(1);
}
```

test-for.catz

```
for(int i = 1; i < 5; i=i+1) {
  print(i);
  print(i*2);
}
```

test-while.catz

```
int x = 1;
while (x < 5) {
  print(x);
      x = x + 1;
}
```

test-if.catz

```
int x=7;
if (x<7){
    print (1);
}
else if (x==7){
    print(8);
}
```

test-return.catz

```
print(my_func());

def int my_func() {
  return 3;
}
```

test-voidReturn.catz

```
print(1);
my_func();
print(3);

def void my_func() {
  return;
}
```

test-createArray.catz

```
int[] x = new int[6];
print(x[3]);
print(x.length);
```

test-printingAnArray.catz

```
int[] x = [5, 3, 6, 4, 7, 3, 2, 1, 6];
for (int i = 0; i < x.length; i = i + 1) {
  print(x[i]);
}
```

test-arrayElementAssign.catz

```
int[] x = [1, 2, 3];
for (int i = 0; i < x.length; i = i + 1) {
  print(x[i]);
}
x[1] = 100;
for (int i = 0; i < x.length; i = i + 1) {
  print(x[i]);
}
```

test-arrayLength.catz

```
int[] x = new int[6];
int[] y = [5, 7, 2, 4, 5];
print(x.length);
print(y.length);
```

test-arrayNotes.catz (more in depth testing in the write tests)

```
Note[] x = [C3, E3, G3, B3];
print(1);
```

test-nestedArrayConstructor.catz

```
int[][] x = new int[][6];
print(x.length);
print(x[0].length);
x[0] = [3, 5, 6];
print(x[0].length);
```

test-nestedArray.catz

```
int[][] x = [[1, 5, 8, 4], [3, 5, 2]];
for (int i = 0; i < x.length; i = i + 1) {
   for (int j = 0; j < x[i].length; j = j + 1) {
     print(x[i][j]);
   }
 }
```

test-include.catz

```
#include "test-if.catz"
#include "test-math.catz"
```

test-randInt.catz

```
seed();
print(randInt(4, 12));
```

test-pitchEquality.catz

```
Note x = new Note(E4, hf);
Note y = new Note(E4, qr);
if (x.pitch == y.pitch) {
  prints("Equal");
}
x.pitch = F#4;
if (x.pitch == y.pitch) {
  prints("Still Equal");
}
```

test-rhythmArray.catz

```
Note[] rhythms = [new Note(C4, wh), new Note(C4, hf), new Note(C4, qr),
        new Note(C4, et), new Note(C4, st), new Note(C4, ts)];

Note n = new Note(C4, wh);
if(n.rhythm == wh) {
      printb(true);
}
```

Fail Tests

fail-doubleBind.catz

```
int x = 5;
int x = 6;
```

fail-undefinedVariable.catz

```
def void foo() {
	print(x);
}
```

fail-voidBind.catz

```
void x;
```

fail-voidBindAssign.catz

```
void x = 3;
```

fail-voidBindParameter.catz

```
def int foo(void x) {
    print("hello");
}
```

fail-conditionalType.catz

```
def void foo(){
    int x = 1;
    if(x){
      prints("hi");
    }
}
foo();
```

fail-illegalAssignment.catz

```
int x;
x = "ECATZ";
```

fail-unary.catz

```
String i = "hi";
```

```
int x = -i;
print(x);
```

fail-illegalBinary.catz

```
String a = "Hello ";
String b = "World";
print(a/b);
```

fail-incorrectParameters.catz

```
int x = 5;
bool y = true;
write(x, y);
```

fail-underfinedFunction.catz

```
hello();
```

fail-redefineFunction.catz

```
int x;
def void foo() {
       x = 5;
}

def void foo() {
       x = 7;
}

foo();
print(x);
```

fail-illegalArrayLiteral.catz

```
int[] arr = [1, 2, 3, "hi"];
print(arr);
```

fail-lengthBadType.catz

```
int x = 5;
print(x.length);
```

fail-illegalBracketAccess.catz

```
String hi = "Hello world";
String bye = hi[2];
```

fail-pitchAccess.catz

```
int[] arr = [1,2,3];
arr.pitch;
```

fail-printPitchAndRhythm.catz

```
Note x = new Note(B@5, qr);
prints(x.pitch);
prints(x.rhythm);
```

fail-badBinop.catz

```
Note x = new Note(E4, hf);
Note y = new Note(E4, qr);
if (x == y) {
  prints("Equal");
}
```

Write Tests

write-noteConstructor.catz

```
Note x = new Note(B@5, qr);
write([x], "write-noteConstructor");
```

write-noteAccess.catz

```
Note x = new Note(D@5, wh);
Note y = new Note(x.pitch, hf);
write([x,y], "write-noteAccess");
```

write-NoteArray.catz

```
Note[] x = [C3, C4, C5];
write(x, "write-NoteArray");
```

write-noteAssign.catz

```
Note x = new Note(B@5, qr);
x.pitch = C#3;
x.rhythm = hf;
write([x], "write-noteAssign");
```

write-reassginNoteFromPitch.catz

```
Note x = new Note(F@4, hf);
```

```
x = C@3;
write([x], "write-reassignNoteFromPitch");
```

write-sequentialNoteAssign.catz

```
Note a = C4;
Note b = C3;
b.pitch = a.pitch = C#5;
write([a, b], "write-sequentialNoteAssign");
```

write-2DNoteArray.catz

```
Note[][] x = [[C3, C4, C5],[A5, A3]];
writeN(x, "write-2DNoteArray");
```

write-2DNoteArrayWithSingleNote.catz

```
Note[][] x = [[C3, C4, C5],[A5, A3],[B2]];
writeN(x, "write-2DNoteArrWithSingleNote");
```

write-integration.catz

```
#include "stdlib.catz"
Note[][] bluesScale = [[C4], [D4], [D#4], [E4], [G4], [A5]];

/* play the scale twice */
Note[][] doubleBluesScale = concatNestedNoteArray(bluesScale, bluesScale);

/* play every other note of the scale */
Note[][] everyOther = new Note[][bluesScale.length];
int j = 0;
for(int i = 0; i < bluesScale.length; i = i+2)
{
    everyOther[j] = bluesScale[i];
    j = j+1;
}

/* create a chord using the notes in the scale */
Note[] cMin = [bluesScale[0][0], bluesScale[2][0], bluesScale[4][0]];
Note[] someOtherChord = [bluesScale[0][0], bluesScale[3][0], bluesScale[5][0]];

for(int k = bluesScale.length/2; k < (everyOther.length); k = k+1)
{
   if(k%2 == 0) {
        everyOther[k] = cMin;
    } else {
        everyOther[k] = someOtherChord;
    }
}

writeN(everyOther, "test-beforeFaster");
makeItFaster(everyOther);
writeN(everyOther, "test-afterFaster");

Note[][] final = concatNestedNoteArray(doubleBluesScale, everyOther);
writeN(final, "write-integration");

/* function that changes the rhythm variable for each Note */
def void makeItFaster(Note[][] arr){
    Note[] rhythms = [new Note(C4, wh), new Note(C4, hf), new Note(C4, qr),
        new Note(C4, et), new Note(C4, st), new Note(C4, ts)];
    for(int i = 0; i < arr.length; i = i+1) {
       for(int k = 0; k < arr[i].length; k = k+1) {
      for(int j = 0; j < rhythms.length; j = j+1) {
          if(arr[i][k].rhythm != ts && arr[i][k].rhythm == rhythms[j].rhythm) {
               arr[i][k].rhythm = rhythms[j+1].rhythm;
            }
          }
        }
    }

}
```

Appendix C:

commit d1a36cc568f9182b3fcb0ece95683733e0aeda08
Author: cina10 <li.energy4@gmail.com>
Date: Mon Apr 26 15:26:01 2021 +0000

    some changes to integration test

commit dcfb529d73b53d733a5c70529f33f29c55c2f5eb
Merge: 9bf1883 6caef5f
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date: Mon Apr 26 11:22:28 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 9bf18839fe15701546f19d837d73bd8b40087034
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date: Mon Apr 26 11:22:15 2021 -0400

```
    quick fix

commit 6caef5f02ed404e17a75313417aa736b92f321cf
Merge: e8ae331 b99ae4c
Author: cina10 <li.energy4@gmail.com>
Date:   Mon Apr 26 14:18:53 2021 +0000

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit e8ae3313a51d76ba3419e39c7844224e5b81e8ca
Author: cina10 <li.energy4@gmail.com>
Date:   Mon Apr 26 14:18:25 2021 +0000

    integration test

commit b99ae4c99f2480f127c4066298ef9304e51ecf1b
Merge: 043b342 0b9bbf5
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 26 09:02:52 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 043b342a15356dc65e774f8e14c35b5ac872ea3b
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 26 09:02:47 2021 -0400

    new and improved rick

commit 0b9bbf5c5d203009e025ecbde6e4f7a105233e9d
Author: Cina10 <li.energy4@gmail.com>
Date:   Mon Apr 26 08:44:50 2021 -0400

    typo in semant + integration test

commit a6e729667e079d98f55a1fd23281b16945e2a58d
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 26 08:17:11 2021 -0400

    changed equality semant

commit 6bc064f4c1132219bf4ffcc412bf6184cabe4839
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 16:41:57 2021 -0400
```

```
    fixed bug with sequential assigns

commit c6b61f09178bfc432057587f1facf25e76ef2bcd
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 15:51:07 2021 -0400

    remove chars

commit b4b05b525de85eb308818a06a48949f5376be13a
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 15:15:39 2021 -0400

    updated stdlib add rick chorus

commit 2ccec2db41a6435bfae7c64310c9d64ad954a6c4
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 14:15:54 2021 -0400

    change serialism starting input

commit b5e07c2f58e3f46f18779a032109e47af26e0598
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 14:08:15 2021 -0400

    utils typo

commit 41424496fee5fd74ab7b1d68d5eff3da4a941a8b
Merge: 274488b c5e9e35
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 13:47:26 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 274488be02a0d6de8d7e9a3f07546eb70b543632
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 13:47:20 2021 -0400

    add rick

commit c5e9e351f24116b789569812d6e84a188b5ea720
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 17:27:22 2021 +0000

    clean and rm write-pitch.catz
```

```
commit c1548aee1dfa26fc37b09bea2fa006c504dd420e
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 17:23:28 2021 +0000

    more correct ly files, (also randInt test changed)

commit 5242295b1c601999a507c67ebeb1b59483d89070
Merge: 6944b5a 9a472e8
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 13:08:57 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 6944b5ae7665d5167c7c78ed2296e9bfd8f543e0
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 13:08:43 2021 -0400

    changes to tests

commit 9a472e86e7476e6a9c9dbaa7076835151979fb72
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 16:31:57 2021 +0000

    correct ly files

commit e42cfb25cdee412ac68bf30d2d983b41c46b8bff
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 12:18:14 2021 -0400

    edit testall

commit a8e1b89f9361fbf1280e13c96f8a9deeda45c1ef
Merge: 42cc941 ca48a48
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 16:06:45 2021 +0000

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit ca48a48abbb1fde78a81635d7011f41a9ae9224e
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 12:06:28 2021 -0400

    oops this one actually changed the shell script

commit 42cc941fef0da4ba25a6f84d613cf3268585c6d0
```

```
Merge: b806011 4600634
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 16:05:17 2021 +0000

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 46006341d1db295cc51c2fc6ec1682f7929d6f66
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 12:04:37 2021 -0400

    change shell script

commit b80601180a5760397fc86861fa806239f45e867f
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 25 16:00:02 2021 +0000

    write2DNoteArray correct file

commit f9d38bb280b1d6f4ba8452f8eba5e2c39e71330d
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 25 10:26:33 2021 -0400

    make run-catz script

commit 7c16c5124582e97d5ec9fc2a7030a3143d3d3fa6
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 21:37:18 2021 -0400

    simplify stdlib and fix semant err

commit d89035f874602bc081a9f928627b122980c4a679
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 20:34:31 2021 -0400

    comments and minor fixes

commit 0b367d2ccefa1739792801abc2a7fb5e8691fcad
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 20:09:54 2021 -0400

    cleanup

commit 977fc351eb4d63001db70fadf0f586c503f30a2f
Merge: 8146fab ce9f4ac
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
```

```
Date:   Sat Apr 24 20:02:23 2021 -0400

    Merge pull request #5 from breadou-sui/harmonystuff

    Harmonystuff

commit ce9f4ac85d3b9a9da321aa8cb0adc54fd7ef2b08
Merge: 1801136 8146fab
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Sat Apr 24 20:02:18 2021 -0400

    Merge branch 'main' into harmonystuff

commit 180113627376dcf1994060148231dcc1e624fde6
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 20:00:30 2021 -0400

    update lilypond syntax. add back old write function and make new one writeN

commit cd28a1f5f562cd50ecd1ea4ea1a646d9e15225cc
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 19:46:00 2021 -0400

    stash utils changes

commit b125e58554d82453a16bab0e85b1e9bee8753d0c
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sat Apr 24 19:04:00 2021 -0400

    need to fix syntax

commit 8146fab5706a6de9084853073b31618b240d0e80
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 24 18:50:47 2021 -0400

    fix random seeding delete array out of bounds test

commit dfa25c93627d001eceaa0c188ecd90693755b181
Author: cina10 <li.energy4@gmail.com>
Date:   Fri Apr 23 13:54:21 2021 +0000

    add fail Note Access

commit 022bd622f5338b5282a291934152b06528e0daac
Author: cina10 <li.energy4@gmail.com>
```

```
Date:   Fri Apr 23 13:40:49 2021 +0000

    check boolean fail test

commit 3f3f5667f70c1637c675f2fdc75f89d360cc4730
Author: cina10 <li.energy4@gmail.com>
Date:   Fri Apr 23 13:06:57 2021 +0000

    oops cleaned up extra files

commit 71e35fcfff84537f3180bc5dc2e4ba0aa9c9ec52
Author: cina10 <li.energy4@gmail.com>
Date:   Fri Apr 23 13:06:10 2021 +0000

    array fail tests

commit 2aa0880fe7030bbb0151e3332ca71471d4ef183b
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 21:53:52 2021 -0400

    fix concat array

commit 7a120ec7094e588e0b3faded102aed893bc30820
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 21:21:58 2021 -0400

    lilypond syntax

commit 0558a4b871a2998ef43a3a43eb0903e50f66cb44
Merge: 5369d68 a1cbb79
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 21:13:42 2021 -0400

    merging:(

commit 5369d686a705f0a7dd996bce1c184db929164f03
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 21:11:41 2021 -0400

    little fixes and cleanup

commit a1cbb79056b2f21f8414147ce98ad7f7364e98fb
Author: root <li.energy4@gmail.com>
Date:   Thu Apr 22 01:09:18 2021 +0000
```

```
    more error tests

commit f11c19ad207f149aaf4cacff60327e0ec1a0be4a
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 20:27:39 2021 -0400

    compiles

commit b2e1b6aa462d5a7de24dd3fa74274a74ce8eaceb
Merge: e1c5dfb b6f4d66
Author: cina10 <li.energy4@gmail.com>
Date:   Thu Apr 22 00:18:55 2021 +0000

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit b6f4d6616fdaf1e69d5498ae37796dd14b6e9973
Merge: 322d301 e179e68
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 20:17:36 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit e1c5dfb8fe9347fee8ba98d3cfff13073c68082b
Author: cina10 <li.energy4@gmail.com>
Date:   Thu Apr 22 00:17:00 2021 +0000

    edit tests

commit 322d3014a2558dec747d7dc6d6957ec42ba8cc41
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 20:15:54 2021 -0400

    write changed a little (getting to array param)

commit fd1b431e9066e3b65a2c54fc4c78077386d7b23c
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 20:10:29 2021 -0400

    include againnn

commit e179e686e96ea7b6f5638bf3237f4ad000e3dd62
Author: cina10 <li.energy4@gmail.com>
Date:   Wed Apr 21 23:31:19 2021 +0000

    fail tests
```

```
commit 7aed83f72790fa45dce135608aff2b21c184b32b
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 16:19:51 2021 -0400

    include worksgit add .

commit ee0f150fe1af57e6ab0e7abaac0670d3741e4851
Merge: 6f5bf8c 2cfafca
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Apr 21 15:38:19 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 2cfafca17269397638e7eb4d323615bac0d6728c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 14:58:26 2021 -0400

    made interesting serialism program

commit 1049953f6171daf266ca31394c8d967c6f664ad8
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 14:50:44 2021 -0400

    updated to stdlib

commit f523b6fefb57eeb67e42513b6cf69e852839ecfd
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 21 14:18:06 2021 -0400

    void binds fails

commit 6f5bf8c912723f7f2f4dafa8ea71dba58967880e
Merge: a5383ea 08cd993
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Apr 20 09:56:52 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 08cd9933753b0d165c2e8e7717d005f75872058c
Merge: 55a869e 748932e
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Mon Apr 19 20:29:55 2021 -0400

    Merge pull request #4 from breadou-sui/id-expressions
    Array modification (all ids are expressions instead of strings now)

commit 748932e832dac8c42061b6768f6a832d061067dc
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 20:22:31 2021 -0400

    array modification?

commit a5383eaad714a99568131efbbfbbc78773bb2690
Merge: ef3103c 55a869e
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 19 19:15:50 2021 -0400

    let me pullll

commit ef3103cf36accec6f37c175e2af7992f6767da92
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 19 18:45:15 2021 -0400

    added some include functionality-unfinished might have to comment out include stuff in scanner

commit c52eb90ba734b2b6fe0b382cd66372889174b2b3
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 13:14:22 2021 -0400

    actually commit what i said before

commit 84291087404f98ab775378053109c3b08f96bd5c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 13:13:57 2021 -0400

    ids are always expressions but is that good?

commit 55a869eb1d2de8594eb20001e8fb1d6d8e9ce5a6
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 10:03:18 2021 -0400

    one last merge conflict + tests

commit fe1ef46d454dddc8b7b6c9b1a3bb10d9ab300a7d
Merge: 07cce27 16ef8ad
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Mon Apr 19 09:58:58 2021 -0400
```

```
    Merge pull request #3 from breadou-sui/array-fun

    Array fun

commit 16ef8ad1903973c2ea15cd2cac24b4adec87f893
Merge: e539a1e 07cce27
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Mon Apr 19 09:58:47 2021 -0400

    Merge branch 'main' into array-fun

commit e539a1ec73b552cf45e076d30e19b2aa16619290
Merge: 3f3c65e db756da
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 09:57:58 2021 -0400

    fix merge conflicts

commit 3f3c65e45370fc0c26f6e5879e8ea7763fee89d3
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 09:50:14 2021 -0400

    arrays work

commit 76f688c3d8806a8be9bc4e416dba80cb2281fa42
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 09:23:29 2021 -0400

    only length doesn't work. no structs

commit e6029d2102f5a917185fa80acd7593f97bfb4cf3
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 19 01:28:40 2021 -0400

    arrays not a struct, but literals work

commit 07cce278280b6cf381f933fd494c18cb05a05c30
Author: cina10 <li.energy4@gmail.com>
Date:   Mon Apr 19 02:41:29 2021 +0000

    Bind void parameter test

commit 80453e4764f8e7996fee774b3e0a4da733ab8b76
Author: cina10 <li.energy4@gmail.com>
Date:   Mon Apr 19 02:41:10 2021 +0000
```

```
    Bind void parameter test

commit f1f1ee2cac9c62d8392e414c3ca4c58630e066e1
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 18 22:36:29 2021 -0400

    a lot of array progress

commit 9a1ca1941ecf4fd41372b5dedfc3ba59af36f09c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 18 17:24:38 2021 -0400

    minor fix in semant

commit 2a6ac65cc49ee234cd1d09f66ecd84ca67f4c17c
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 18 20:43:33 2021 +0000

    cleaned up directory

commit 26808643b7231b464f28d55b71201fdf07aa3eb4
Author: cina10 <li.energy4@gmail.com>
Date:   Sun Apr 18 20:43:14 2021 +0000

    string issue resolved

commit 3acaaa78c15dafc733c6f2e9d52887984b0c67f1
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 18 10:32:03 2021 -0400

    finish ast and semant for arrays

commit db756dae27c6c453fad4778aab9a587e5a4aa56b
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Fri Apr 16 13:53:42 2021 -0400

    add a fail test

commit 18ce146ddcbd79820a30c79414417520aeb958ed
Merge: e719a75 1f9872e
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Fri Apr 16 13:50:17 2021 -0400

    Merge pull request #2 from breadou-sui/operations_on_notes
    stdlib, fix main, cleanup, precedence

commit 1f9872ee0116a419875feb03f54a418f089b2974
Merge: 2536e21 e719a75
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Fri Apr 16 13:50:06 2021 -0400

    Merge branch 'main' into operations_on_notes

commit e719a757e67e9157a86b9644d71c505c595d3288
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Fri Apr 16 13:48:10 2021 -0400

    main compiles without warnings. arrays raise implementation error

commit 2536e211e01ff66c731fe070b4e073e62e86d1c3
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Fri Apr 16 13:42:13 2021 -0400

    beginnings of stdlib + make main return 0 instead of void

commit 2fb64b7fb5a16ede361d7622a83f3296335c3017
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Thu Apr 15 00:16:27 2021 -0400

    beginnings of potential stdlib and some tests

commit 8dc0610ff6339ef791452ca236bb1ad20950b4f9
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 14 23:16:24 2021 -0400

    fixed the weirder string issue but still always has the quotes as part of it

commit cd32184b276fe46e6a1c6971f6f31a1d292cb4a1
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Apr 14 23:01:26 2021 -0400

    hardcore arrays

commit 42aeae86c5ada6da58868f0e024d60f9c0f2d4b1
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Wed Apr 14 22:09:29 2021 -0400

    writing arraylength
```

```
commit 0abf1043d87a4b52ee90321acf82ed2119b3eaa0
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Wed Apr 14 21:20:09 2021 -0400

    testing array parser.mly

commit e92f76d3f07f96a57dd5155e8dbe9c3e3074bcbb
Merge: 9311691 e96cb8d
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Apr 13 19:28:18 2021 -0400

    if/else statements and unops neg/not +tests

commit 9311691115e15a1876ff3b62e51f7f47db18949e
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Apr 13 19:26:49 2021 -0400

    if/else and unary ops neg/not +tests

commit d503e9c0dd04919d0b0be634620609acde542cd0
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 18:40:15 2021 -0400

    add precendece

commit 2a4545ff24129d11cc33b5982a934abb3ec6aa2b
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 18:36:37 2021 -0400

    combined redundant methods

commit e96cb8db848660ca71123ec2717b87e3dd682e8c
Merge: 8a7aeda c003478
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Tue Apr 13 12:44:01 2021 -0400

    Merge PR: basic write, rhythms, note structs, dot access

    absolutely not fully tested and i think there are a lot of potential unchecked fail states

commit c00347849d7243c974f902f5bd1571c7ec27a18d
Merge: 9a66fe3 8a7aeda
Author: Tim Vallancourt <41602450+TPVallancourt@users.noreply.github.com>
Date:   Tue Apr 13 12:43:14 2021 -0400
```

```
    Merge branch 'main' into write_func_real

commit 9a66fe3ef5f783092e73857ad8b8d98afd597a7d
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 12:39:45 2021 -0400

    added ability to change individual elements of note

commit 13295d731068d2291e1d943aa95e5137ec3ef507
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 12:26:49 2021 -0400

    fix dot access

commit 5e30f101bc388e9fc5033c396d7a189051357176
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 12:11:00 2021 -0400

    started dot access; currently segfault

commit 2e828791ef618dfd4abcfac3e7f42dade6b57601
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 13 01:21:35 2021 -0400

    add rhythm literals, make note a pitch rhythm struct, write rhythms to lilypond

commit 8a7aeda682d0e58742841fcaf031394f1f28cd23
Author: cina10 <li.energy4@gmail.com>
Date:   Tue Apr 13 02:51:07 2021 +0000

    some array stuff in the parser

commit aa2347f8e80d0b9b17453321ae49a3687bb3e73e
Author: cina10 <li.energy4@gmail.com>
Date:   Tue Apr 13 02:24:44 2021 +0000

    parser changes (and cleaned up files

commit a3bb6a5a16f26c291283b19bdc3a1210cdd55824
Author: cina10 <li.energy4@gmail.com>
Date:   Tue Apr 13 02:23:10 2021 +0000

    array changes
```

```
commit 321ed9b7d8ef230492678630561e2d7e49614b1c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 12 21:05:33 2021 -0400

    write function takes in single note + filename and makes lilypond file

commit 2fd6385c75e7372f4200acbf94dcaecfbbd5cf54
Merge: 88a349f 29d9cca
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 12 14:56:47 2021 -0400

    merge conflicts resolved +if/else/not/neg

commit 88a349f25c32d301f169b2eae871a801969840cf
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 12 14:52:18 2021 -0400

    DS Store file

commit 7735d66980c2d934a5c6c361d36aa66d3406b1f8
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 12 14:49:03 2021 -0400

    literally no change

commit 272d01e8b2722515f3a5051629cf21618d7d9239
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Apr 12 14:47:32 2021 -0400

    If  statements, not/neg

commit 29d9ccae6b17f8a8d40a333d2bf95c966bfbc526
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 12 11:39:15 2021 -0400

    ast prints out forwards

commit 23c083f493e7689967f35b8c97a152a0338bdfb8
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 12 10:04:51 2021 -0400

    directory cleanup

commit 36cab418e3cb53c5b93da7c5752c79a6f5902c5c
Author: Tim Vallancourt <tpv2106@columbia.edu>
```

```
Date:   Mon Apr 12 09:53:43 2021 -0400

    notes are just pitches are just strings

commit 1b59ce7de8605760e5f2052f819bf27ccf7cdbab
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Apr 12 09:01:14 2021 -0400

    for loops working. sliteral warning gone

commit a0f66baa7c4e05df5328d745d3d3dc3cea9b9954
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 11 22:13:24 2021 -0400

    add comments

commit 98c58890e36f658cc13f071d855aebac0d6c5c80
Merge: 1c7b77c c665180
Author: cin10 <li.energy4@gmail.com>
Date:   Mon Apr 12 01:10:34 2021 +0000

    test-while

commit 1c7b77cef55491a6bb7c9b5602d3a46d353810d8
Author: cin10 <li.energy4@gmail.com>
Date:   Mon Apr 12 01:01:24 2021 +0000

    change to test while

commit c66518092eccbd47015cc99cb8a8ecc4a3fcba13
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:59:59 2021 -0400

    change numbers

commit b4c1d8254898edc55fc24b549f7b0036631127cc
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:40:18 2021 -0400

    test.out files

commit b0347f5d6af0557a4fc616d836888c0300beadbe
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:33:03 2021 -0400
```

```
    rename test file

commit 9c036802945730cee1d91260959ed551cab10400
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:31:38 2021 -0400

    test files and for loop

commit bd50ed0bd134074de756d2da90e11728741b4412
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:22:22 2021 -0400

    binary  printing

commit 87dafdca44df0fba56bec91eb4ff35fc7e62cb0a
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:18:12 2021 -0400

    edit parser

commit 26d123ab3e38c0b47d9ccf830b16186ebb8be752
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 20:17:37 2021 -0400

    edited parser

commit 64c5d549f14763f67e7ae586317748cf3a07729e
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 11 20:00:04 2021 -0400

    loops can loop on one thing

commit 315307d449fdde56254618f37f9240b41d0e0a16
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 19:37:10 2021 -0400

    loop things

commit d7f23144a540aa726bd6e5a5f459d492abfb90c4
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 19:34:11 2021 -0400

    loops stuff

commit 462ab986568a12c8979aeb6cd7a3a9f468d42be9
```

```
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 11 19:23:41 2021 -0400

    strings can be printed

commit c038846c5934633d33c10f011b599fa9ec3c466d
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 19:11:33 2021 -0400

    rearranged again

commit 7c83753fbf3dd5c239f93d364ede0a4ca0ec83b6
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:54:38 2021 -0400

    test old file

commit 6d57b530bacfe1c6ba09ed4b1c29957293c3c2d5
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:53:04 2021 -0400

    not sure if  the last commit worked

commit 9205084481915bdbeff1c0857d2d8ca1b4cd5d35
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:46:29 2021 -0400

    try moving again

commit 0767f94ec653ff66ea1b5343637a695abde2ddf7
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:13:57 2021 -0400

    testall?

commit 90773be3eedef27cb881526c82806e222e49c2e2
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:12:19 2021 -0400

    testall?

commit 644c77da9139b5e917aa7b0a02dcd9fedc7ca7e5
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 18:11:10 2021 -0400
```

```
    removed duplicate builder

commit a3b4dab4817277285e3d5ba34d64dd51e6953527
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 17:50:39 2021 -0400

    editing print calls

commit ec4537625ee169e6ff447cd1828adec4a6716ea5
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 17:46:09 2021 -0400

    more moving around print calls

commit 0c9256d894c35df60288561dcba8f55637a303c3
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Apr 11 17:38:34 2021 -0400

    move (builder,_) below build_stmt

commit ec2b683baf54e1dcae0942f6881dd9e2ff02b662
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 16:07:34 2021 -0400

    rearranging code

commit ac2cc6aee7d49031334f75d5cc3ccfac879ed4de
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 16:03:29 2021 -0400

    added print pointers

commit 7c21daefb6942de86cfd8fe35d65f581cc1e08d0
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 15:53:59 2021 -0400

    added more print functions

commit 780fc4f574d061aa087ddaab483e583e845f7b72
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 14:58:16 2021 -0400

    testing b print

commit 77a394d4aa75cfb0da70a4b31bbd0bccd7bd824a
```

```
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 14:54:42 2021 -0400

    added constructor for sliteral in parser.mly

commit 2aa636c299b1d72ca360df8e26c9d70e8947ddf1
Merge: 0900bf5 415cf61
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 14:46:29 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 0900bf5d167f23f83c91435df38fea43df836561
Author: Annie Sui <squishyrainbow@gmail.com>
Date:   Sun Apr 11 14:46:08 2021 -0400

    testing strings

commit 415cf61a7b3ab45c6bdc96d41dbc9b2981ff0c48
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 10 15:42:12 2021 -0400

    tweaks

commit c8729d2213a2904901edd5d5f93bae3f38586da2
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Fri Apr 9 11:49:58 2021 -0400

    creates a locals table that gets passed around. allows for assignments. needs to be tested

commit 251e34fe67009773183e3fc2dd1d1ce56a5e581c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Thu Apr 8 12:52:29 2021 -0400

    changed stmt building so there isn't duplicate code

commit 777bf035f3dec18750aab7cf7e849dfdaba6c2f4
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 6 19:00:31 2021 -0400

    make randInt a 2arg func

commit 2dc9f96cbf97485fa1a1238151a8db76c198af1e
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 6 18:18:15 2021 -0400
```

```
    cleaned up test directory. testall broken?

commit b333191baef76976b1ac3a9740d7979d824f7466
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Apr 6 13:50:35 2021 -0400

    retool how checking statements in functions works. not my favorite. added return statements

commit 63ac30a2b1f22cf24e3ee8b6b95d5e55ab958ad0
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Apr 4 23:58:04 2021 -0400

    add randInt and change name of c file to utils

commit 82369e20455dccecbb50c7b2d8a832df3dce3804
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Apr 3 17:18:06 2021 -0400

    no more main method:)

commit 893a17ea768dfaec71dab0ab0d9da5319c796308
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Fri Apr 2 17:15:02 2021 -0400

    add bool and float literals

commit a571d8438d0ff1b7ccbffd5034ef6917a15d7c45
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 29 16:34:28 2021 -0400

    added printf for testing; fixed warnings

commit 65458fe1c5c855bca2bfec2f1bb9ad103e62dd72
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Mar 28 20:14:09 2021 -0400

    math:)

commit 6bb8953e28f32c13c4a98e79ab6ba693f3cf50f2
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Mar 28 19:39:54 2021 -0400

    added the real folder that we'll work on
```

```
commit 2f0d5a7117c52f41acf288d8bc85f0655ec5ade5
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Mar 24 19:40:03 2021 -0400

     created tar.gz file

commit 16d91fca93098fdf5056da901c1acfaea9e4c31b
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Wed Mar 24 19:36:27 2021 -0400

    added readme

commit 7a52e876643b5585909712d63ba1acaf45291838
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Mar 24 18:21:17 2021 -0400

    i have committed terrible terrible crimes but its over its over its over

commit 06a683daf72d094ea73696a9114323d2caf8852a
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Mar 24 13:18:51 2021 -0400

    fix warnings outside of codegen

commit d0fa80f0c11a221313f87e165423ea8214fd2294
Merge: e8edb94 a6ecb40
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 23:55:37 2021 -0400

    Merge branch 'main' of github.com:breadou-sui/ecatz into main

commit e8edb94d82056b5afe68d624be843bf1c4a00d67
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 23:54:20 2021 -0400

    Merge branch 'main' of github.com:breadou-sui/ecatz into main

commit fda31c301be054624b08ebe6d6d94c694445e1a8
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 23:51:45 2021 -0400

    add pretty printing; fix semant? codegen broken

commit a6ecb4023591a8c511848847c7baf797e0f5b8e5
Merge: 7f52d85 a2650c4
```

```
Author: Cina10 <li.energy4@gmail.com>
Date:   Tue Mar 23 22:44:36 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 7f52d850b9b17b131f094b9b2a4340b13f97071e
Author: Cina10 <li.energy4@gmail.com>
Date:   Tue Mar 23 22:44:28 2021 -0400

    add write function in c

commit a2650c4d591e93a3919e57b8bb7c14f1adcb5c64
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 22:19:23 2021 -0400

    semant is a terrible beast and it does not work

commit 4f42d49d24907906a7f8c1804b8ce45c50fe27a7
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 22:04:21 2021 -0400

    at least it fucking compiles

commit c5934b444a7164707cfbdc7c35edf8f27402a5b9
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 21:25:51 2021 -0400

    new fixes

commit 28e1c75ec08d1a0f47321d4fbae5688107648a3f
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 20:00:49 2021 -0400

    getting real errors now hell yeah

commit a7dadd2403aa7ce289813eebe9833028ddd8f8cf
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 19:46:51 2021 -0400

    make tags say the right thing :/

commit 303561bfc2f2b22f9cd8ec1fc6bf47c34fd82b73
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 19:45:39 2021 -0400
```

```
    add tags file and delete extra in in semant

commit 1bb26bfcb136989139114e35b964d8cde95b0c1e
Merge: 38b590f 8bd5056
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 18:16:57 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 8bd5056e7e061b148f3d8bf5de4712c61f61b155
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 18:16:42 2021 -0400

    add hello world makefile

commit 38b590f30659ee96cc2a15ef48aff4b298e2c561
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 18:03:35 2021 -0400

    added microc.ml file equivalent

commit f75caa34062d51dfa149fcc9954faedc5356d99e
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 17:56:47 2021 -0400

    maybe we finished semant??

commit 0475d6ed652c249b2ec43ecff4aefbe22729f5da
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 17:30:06 2021 -0400

    tenatively finished codegen??

commit fb9a5694ba39d75cce463fb3d273c1c58cb682fc
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 16:56:18 2021 -0400

    fill in note struct arguments

commit ec5ed45ccaaf9f8577157345d6dedd4fdbbf5b5f
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 16:00:38 2021 -0400

    deleted output/duplicate files
```

```
commit 8dd65d9ee5055edf524297bfa9dd52a11e70270f
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Tue Mar 23 15:52:17 2021 -0400

    updated first four files to better reflect microc func calls

commit 5730d0834c5837d2f3caea4fb143076448e2379c
Merge: d639efd 2809509
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 12:37:37 2021 -0400

    added to codegen and sast

commit d639efda34819e852748b0e052cfca9ba0ddf6bb
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Mar 23 11:37:54 2021 -0400

    commiting to merge

commit 2809509ad080c9216e0de7d040f67be44400d32f
Author: Annie Sui <breadou@Annies-MacBook-Pro.local>
Date:   Tue Mar 23 11:36:26 2021 -0400

    hello-world updated files for 3/23 meeting

commit db61c24343d9f41dbc1b127bdefdba8abb6bd198
Author: Annie Sui <breadou@Annies-MacBook-Pro.local>
Date:   Sun Mar 21 19:22:10 2021 -0400

    testing hello-world

commit 0701783215e22f2ca3d65927466b144dfd7b8121
Author: Annie Sui <breadou@Annies-MacBook-Pro.local>
Date:   Wed Mar 17 17:30:32 2021 -0400

    worked on ast with Chianna, left off at assign()

commit a6cc05e0c2fc8642a696a68ec95510c9c5d6ea26
Author: Cina10 <li.energy4@gmail.com>
Date:   Wed Mar 17 15:23:02 2021 -0400

    small changes, add some microC

commit 79c21ec4b2c6f5ef339ce285b4c23fd9916037c2
Author: Tim Vallancourt <tpv2106@columbia.edu>
```

```
Date:   Mon Mar 15 21:47:41 2021 -0400

    introduce rhythm literals and fix make clean

commit 412deca9fe63df965553589067be4b5f560bc863
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 15 20:02:44 2021 -0400

    change noteArgs

commit 5c9116370faad68bf5a11271abc4e54283fc6819
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 15 19:48:11 2021 -0400

    took some guesses at the braces

commit f2329eb3e83ae3d3dacbcdbf6fb64d880e98e172
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Mar 15 16:32:23 2021 -0400

    filled in brackets

commit e407d09507bda14494e1ef55f1686b5190f0166c
Merge: 60bc7a0 97e1b2c
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Mar 15 15:10:09 2021 -0400

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 60bc7a06e05dc8beaf0fff30f292a1ebc77f58f5
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Mar 15 15:09:53 2021 -0400

    small changes

commit 97e1b2c29a5787e838a9a355ddc0680c60fa8b51
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 15 12:32:07 2021 -0400

    moved control flow into statements

commit df90faba30ccd545e2361c30fe5be3ef54b6c114
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 15 12:29:24 2021 -0400
```

```
    updated parser

commit 5aeeb8d4b86e7993bff8b8be11d640b9b455eeae
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Mar 15 11:25:24 2021 -0400

    make clean and update scanner to be more like microc

commit 8fc9fff2489ab7c3245fd1a7032342931f7e6c55
Author: Cina10 <li.energy4@gmail.com>
Date:   Sun Mar 14 19:44:51 2021 -0400

    parser adjustments

commit 1d4f5823f601020eba6ab58287922d0568761561
Author: Cina10 <li.energy4@gmail.com>
Date:   Wed Feb 24 15:32:05 2021 -0500

    add include key word

commit d807270870fa05b5985a0e773e7216c5b5d54792
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 14:07:19 2021 -0500

    fix bool_expr again because i messed it up

commit c32b6b4ccd2a14920179dcf4d9f22ead4581a9da
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 11:32:39 2021 -0500

    change outputs of bool_expr

commit 3fe076363d88508c2f4bca224dc2637fd5298386
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 11:29:13 2021 -0500

    fix dangling commas

commit 4881d04dadbf66fd10c3b0fa8cdf45a1281aa102
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 11:20:28 2021 -0500

    very basic makefile

commit 763034d908f68202d8414e7d565ba52c671dee52
```

```
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 11:10:05 2021 -0500

    clean literals section in scanner

commit 63a8b04eaeb2fed302dd90e75f86d57ed38bf297
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Wed Feb 24 11:00:32 2021 -0500

    removed output files

commit 9fa25ff03d6d7cf8741d9f5d0a6442c984ea6993
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Tue Feb 23 18:12:17 2021 -0500

    added char lits

commit 26b257a038eeca4b930d88e865ffa4b0043cc224
Merge: 4606c22 40d9872
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Feb 22 21:55:53 2021 -0500

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 4606c22fd973635643cd9eb16d06824693cd3071
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Mon Feb 22 21:55:50 2021 -0500

    committing so i can pull

commit 40d9872b46c60642399ecf5dc69a200a4ed0cd15
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Feb 22 15:08:58 2021 -0500

    add negative numbers

commit fe1844843ac62215021471995d5c1ac4e575e112
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Feb 22 13:44:59 2021 -0500

    modification to note arguments

commit 662433b86364c92d33baebf6db67b9b178b3c2a7
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Mon Feb 22 12:52:26 2021 -0500
```

```
    fix array typing issue

commit d634400bf7177e9792c55ac166a3256bbd7b5c7a
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Feb 21 19:35:48 2021 -0500

    2/21 night meeting

commit 8e7190177c1a503d7921ea974a1130dc138da057
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Feb 21 16:21:02 2021 -0500

    add floats

commit 68095db094dd57448d2a523b23da2b24b1ddc92c
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Feb 21 16:02:29 2021 -0500

    modified associativity and how bool_expr works

commit 112ea71d0437d855f1692ffcd53e4a5285ac23fb
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sun Feb 21 15:17:27 2021 -0500

    restructured parser, clarified stmt v expr

commit 73c8fc2850122929f54e9595c551c717a0415ec4
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sun Feb 21 13:20:37 2021 -0500

    compiles trying to reduce rules never reduced

commit 1f230f44ec6ca88ebd363f23ae6a1aabe349fe3b
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sun Feb 21 12:49:56 2021 -0500

    no compile errors

commit 7db7183c0b825c827054889b12b726c26f5b6271
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sun Feb 21 12:21:59 2021 -0500

    2/21 meeting changes
```

```
commit 333c2a85ac0f7a68f375a348581c61476960541f
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sat Feb 20 22:59:14 2021 -0500

    p sure i just added double? maybe did something else i forget

commit c94ca39f95952e44b1f6eca49645291c90f9248f
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sat Feb 20 22:58:43 2021 -0500

    added all the tokens? (i think) built out more types (built in expressions, expr, punctuation, stdlib (idk
maybe this is how we do this), rhythms (also could be wrong)

commit 57b6bf5296a0991fa55b43d297a0407e8a3840b3
Author: ethiopia mengesha <ethiopia@ethiopias-MacBook-Pro.local>
Date:   Sat Feb 20 22:57:03 2021 -0500

    added some expr function defs

commit 91b8d271639dec9ad8c1864c2fb918339e99b42a
Author: Tim Vallancourt <tpv2106@columbia.edu>
Date:   Sat Feb 20 20:47:37 2021 -0500

    added rhythms and true false to scanner

commit 0766a81bccb8dd6bfe1ccfc23b422f532be5bfa7
Merge: 95a2183 3cab0ba
Author: Chianna Cohen <chiannacohen@Chiannas-MBP.fios-router.home>
Date:   Fri Feb 19 14:07:38 2021 -0500

    Merge branch 'main' of https://github.com/breadou-sui/ecatz into main

commit 95a2183b290bf68ba94a2dcd7da0f1487adea9da
Author: Cin10 <li.energy4@gmail.com>
Date:   Fri Feb 19 13:57:54 2021 -0500

    beginning of parser

commit 3cab0baa63fcdf846f8547603f5131a5f64a40ae
Author: Chianna Cohen <chiannacohen@Chiannas-MBP.fios-router.home>
Date:   Fri Feb 19 13:57:54 2021 -0500

    beginning of parser

commit da4ef9fc6916c89ec5e0884de10e1bf386f3c5fc
```

```
Author: Chianna Cohen <chiannacohen@Chiannas-MBP.fios-router.home>
Date:   Fri Feb 19 13:07:03 2021 -0500

    scanner

commit c863ddf673a44c8ac45cdae235226561c0b2d655
Author: Annie Sui <46787174+breadou-sui@users.noreply.github.com>
Date:   Sun Feb 14 19:11:12 2021 -0500

    added template parser code from hw1 q3

commit b4a7a9346430bf13632179a54dd0eb8387899b54
Author: Annie Sui <46787174+breadou-sui@users.noreply.github.com>
Date:   Sun Feb 14 19:03:40 2021 -0500

    Initial commit
```

Appendix D:

rick.catz

```
#include "stdlib.catz"

Note[] octb = [new Note(B@1, et), B@2];
Note[] octa = [new Note(A1, et), A2];
Note[] bfmaj = [B@3, B@1, D4, F4];
Note[] cmaj = [C4, A1, E4, G4];
Note[] dmin = [D4, F4, A4];
Note[] dmint = [D4, F4, A3];

Note[][] intro = [bfmaj, octb, cmaj, [C4], cmaj, octa, dmin, [new Note(C5,
st)],
                  [new Note(B@4, st)], [new Note(A4, et)], bfmaj, octb,
cmaj, [C4],
                  [new Note(C4, et), E4, A1], concatArray(octa, [F4]),
dmint];

Note[] verse = [D4, E4, F4, F4, G4, E4, D4, C4, D4, D4, E4, F4, D4, C4, C5,
C5, G4];

for (int i = 0; i < verse.length; i = i + 1) {
  if (i != 12 && i != 14) {
    verse[i].rhythm = et;
  }
}
verse[7].rhythm = wh;
verse[16].rhythm = hf;

Note[] cmaje = [new Note(C4, et), A1, E4, G4];
Note[] dmine = [new Note(D4, et), F4, A4];
Note[] spacer = [new Note(B9, st)];
Note[] spaceret = [new Note(B9, et)];

Note[][] neverGonna = [[new Note(C4, st)],[new Note(D4, st)],[new Note(F4,
st)],[new Note(D4, st)]];
Note[][] giveYouUp = [dmine, spacer, dmine, spacer, cmaj, spaceret];
Note[][] letYouDown = [cmaje, spacer, cmaje, spacer, dmint, spaceret];
Note[][] runAround = [bfmaj, [new Note(G4, et)], [new Note(C4, hf), E4],
[new Note(C4, et)], [A3, C4, G4], dmint];
Note[][] sayGoodbye = [[C4, C5],[new Note(E4, et)], [F4], [new Note(E4,
et)]];


Note[][] chorus = new Note[][0];
for (int j = 1; j <= 12; j = j + 1) {
  if (j % 2 == 0) {
    if (j % 6 == 0) {
      chorus = concatNestedNoteArray(chorus, runAround);
    } else if (j == 2 || j == 8) {
      chorus = concatNestedNoteArray(chorus, giveYouUp);
    } else if (j == 4) {
      chorus = concatNestedNoteArray(chorus, letYouDown);
    } else {
      chorus = concatNestedNoteArray(chorus, sayGoodbye);
    }
  } else {
    chorus = concatNestedNoteArray(chorus, neverGonna);
  }
}

writeN(intro, "rick-intro");
write(verse, "rick-verse");
writeN(chorus, "rick-chorus");
```

serialism.catz

```
#include "stdlib.catz"

/*
This is an implementation of serialism, allowing one to randomly
generate masterpieces of early 20th-century music
*/

/*
return a new array that is the notes of an input array
transposed up or down by n
*/
def Note[] transpose(Note[] input, int n) {
 Note[] output = new Note[input.length];
 for (int i = 0; i < input.length; i = i + 1) {
   output[i] = addToNote(input[i], n);
 }
 return output;
}

/* return a new array that is the reverse of the input array */
def Note[] retrograde(Note[] input) {
 int l = input.length;
 Note[] output = new Note[l];
 for (int i = 0; i < l; i = i + 1) {
   output[i] = new Note(input[l-1-i].pitch, input[l-1-i].rhythm);
 }
 return output;
}

/* 'invert' the array by flipping the distances between each note */
def Note[] inverse(Note[] input) {
 Note[] output = new Note[input.length];
 output[0] = input[0];
 int initialPitch = noteToInt(input[0]);
 for (int i = 1; i < input.length; i = i + 1) {
   int delta = noteToInt(input[i-1]) - noteToInt(input[i]);
   Note newPitch = addToNote(output[i-1], delta);
   output[i] = new Note(newPitch.pitch, input[i].rhythm);
 }
 return output;
}

/* seed the random number generation*/
seed();

Note[] finalOutput = new Note[0];
Note[] input = [C4, E4, G4, A4, B4, B4];
for (int i = 0; i < input.length; i = i + 1) {
  input[i].rhythm = et;
}
finalOutput = concatArray(finalOutput, input);

/* randomly choose a function 25 times, concatenate the results to output
*/
for (int i = 0; i < 25; i = i + 1) {
 int randomInt = randInt(0, 2);
 if (randomInt == 0) {
   input = transpose(input, randInt(-12, 12));
   finalOutput = concatArray(finalOutput, input);
 }
 else if (randomInt == 1) {
   input = retrograde(input);
   finalOutput = concatArray(finalOutput, input);
 }
 else {
   input = inverse(input);
   finalOutput = concatArray(finalOutput, input);
 }
}

write(finalOutput, "serialism");
```