Beathoven Final Report

Submitted by:

Eunice Kokor (eek2138), Jake Kwon (jk3655), Rodrigo Manubens (rsm2165), Ruonan Xu (rx2135), Sona Roy (sbr2146)

December 21, 2016

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Beathoven

Our language creates songs phrase by phrase. This allows users to modify entire sequences of notes by either using our standard library or through user defined functions of their own. Additionally, since our language does create MIDI files, having this internal structure provides a closer mapping to the format of MIDI files. This allows the user to take full advantage of what MIDI files can musically describe and represent.

1.2 Goals and Uses

- Our language's output creates a MusicXML or MIDI file representing a music score. We chose this output because it can be imported into various music software programs, such as MuseScore (a free music composition software, which can generate score and play the notes). Additionally, we also chose it because its structure allowed us more flexibility in creating our language's basic functionality.
- One of our stretch goals with this language is to create a music file that contains both melody and lyrics. Lyrics will not have tone, but rather have a beat. The best musical genre for this language will be rap.
- Another goal is to represent chords, notes, and improvisation such that different types of music creators (people who create music by relative pitch vs by absolute pitch) can have more flexibility.
- Another one of our goals with this language will be to easily generate "stacked" music scores, aka music that is played at the same time with the same key but that have different (ie polyphonic) melodies.

Chapter 2

Language Tutorial

2.1 Environment Setup

2.1.1 Installation Under OSX

1. Install Homebrew

```
ruby -e "$(curl -fsSL
    https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Install and setup opam

```
brew install opam
opam init
```

3. Install llvm

```
brew install homebrew/versions/llvm38
```

4. Have opam setup your environment

```
eval `opam config env`
```

5. Install Ocaml Libraries

```
opam install core
opam install llvm.3.8
opam install yojson
```

6. Create a symbolic link to the lli command

```
sudo ln -s /usr/local/opt/llvm38/bin/lli-3.8 /usr/bin/lli
```

2.2 Usage

2.2.1 Using the Compiler

Inside the directory 'Beathoven' type make. This creates the beathoven compiler that takes in '.ml' files and compiles them to corresponding '.ll' files corresponding to LLVM IR. The syntax for running the dice executable is:

```
./beathoven.sh < testfile.bt > outputfile.bt
```

Where the input test file is piped in, but could also be piped out into an output file to see the resultant LLVM. However, if you want to test directly inside the terminal with written code, you can use:

```
./beathoven.sh
```

And write out the desired code. End your statements with a ";" separator, then press "Ctrl + D" to exit and run the code. The output LLVM code will be generated in the terminal.

2.2.2 MIDI File Generation

Once make is run, Beathoven internally creates the midi file specified in the source code. The midi file can be found in the bet_midi_library directory

Chapter 3

Language Reference Manual

3.1 Beathoven

Digital Music Production has become a very powerful tool for all kinds of musicians in this day and age. Through technologies like MIDI, musicians can experiment with and create multi-track compositions. With the appropriate software one can recreate any type of music, be it a guitar riff or an orchestral symphony.

Beathoven is a programming language that generates MIDI/MusicXML files so that anyone, even people who don't really know music, can compose songs by putting together words that represent musical concepts.

Our language creates songs phrase by phrase. This allows users to modify entire sequences of notes by either using our standard library or through user defined functions of their own. Additionally, since our language does create MIDI files, having this internal structure provides a closer mapping to the format of MIDI files. This allows the user to take full advantage of what MIDI files can musically describe and represent.

3.2 Data Types

3.2.1 Basic Types

`Beathoven` has a variety of fundamental data types such as `bool`, `int`, `double`, `char`, `string`.

There are two basic types `pitch` and `duration` that are specific to the music language.

All basic data types are passing by value. The other data types, such as `Struct` and `Array` are passing by reference.

pitch

Beathoven has two types of syntax for pitch values.

- Absolute Pitch: For absolute pitch, the accepted pitches are ['A'-'G'] (['0'-'9'] ('#'|'b')?)?.

  ```
  pitch p1 = C3#;
  pitch p2 = D5;
  pitch p3 = E;
  ```

- Pitch relative to key: For relative pitch, the accepted pitches are ([1-7][^|_]?).

  ```
  pitch re = 2;  /* equivalent to D4 */
  pitch fa = 4^; /* equivalent to F5 */
  pitch la = 6_; /* equivalent to A3 */
  ```

Rest is denoted by a silent pitch type defined as 0.

```
pitch rest = 0;
```

duration

duration is the length of time that a note is played. The whole note has a duration of 1, and the half note has a duration of 1/2, etc.

```
duration quarter = 1/4;
duration quarter = 2/4;
duration quarter = 1/1;
```

Note

A Note is internally defined as a struct. It has a pitch and a duration, which can be accessed with note.p and note.d.

```
Struct Note {
   pitch p;
   duration d;
}
```

There are several ways to define a Note's value, using a pitch, a duration, or both (a pitch and a duration concatenated with ..). The default pitch of a note is C4. The default duration of a note is a quarter note 1/4.

```
pitch p = F4#;
duration d = 1/16;
Note fSharpShort = p..d;
Note fa = 4..1/4;
Note defaultF = p; /* F4# pitch, 1/4 duration */
Note cWhole = ..1/1; /* C pitch, 1 duration */
```

3.2.2 Struct

Beathoven supports user-defined `structs`, which can contain any data types.

```
Struct I_am_a_struct {
   int val;
}
Struct I_am_a_struct an_instance;
```

Right now, there is one built-in struct defined by `Beathoven`, i.e. `Note`.

3.2.3 Array

Arrays are homogeneous. An array can hold multiple elements of the same data type, which can be either primitive or non-primitive. Arrays are 0-indexed and are specified by square brackets `[]`.

```
int[] arr = [0, 1, 2, 3, 4, 5, 6, 7];
pitch[] key = [C, D, E, F, G, A, B];
Struct I_am_a_struct [] struct_array = [];
```

The space for array is allocated dynamically. There is no need to specify its size. Arrays are mutable and its elements and subsequence can be easily accessed like in python.

Index

Element of array can be accessed using `array[idx]`.

```
arr[0] = -1;
```

Subarray

`array[idx1:idx2]` gives a copy of the original array from `idx1` to `idx2` (excluded). In other word, `Beathoven` uses a Pythonic way.

```
arr[:5]; // returns an array of 5 elements from 0 to 4 in arr
arr[1:7]; // returns an subarray of 6 elements
arr[1:]; // returns an subarray of elements from 1 to the end
arr[:]; // returns a copy of the original array.
int[] arr_alias = arr;
int[] arr_copy = arr[:];
```

Concatenation

All arrays have only one dimension. Arrays within brackets [] are flattend .

```
int[] arr_concat = [arr, 8, 9, arr[1:6], 10];
```

The above example get a new one-dimention `int` array by using [] to concatenate an array `arr`, an `8`, a `9`, a subarray, and a `10`.

3.2.4 Seq

A `Seq` is made up of `Note`s or `Chord`s (not yet supported, so a `Seq` is equivalent to a `Note` array at this implementation).

`Seq` is a special case of arrays. all array operations can be applied to a `Seq`.

There is also a special syntax for simple `Seq` declarations.

Elements within brackets `<>` are separated by space instead of comma. Supported element types are `int` (which can also be interpreted as `pitch` musically), `pitch`, `duration`, and `Note`. Note that a `Note` must be in a complete form, i.e. `Note ..1/1` (missing pitch) is not allowed.

```
Seq seq1 = <5..1/1 5 ..1/1>; // two G4 whole notes
Seq seq2 = <5..1/1 B5b..1/4 1 p note C
      p..1/2  p..h 1 ..h 7..h F..h >;
Seq twinkle = <1 1 5 5 6 6 5..3/8>;
```

A "twinkle, twinkle, little star" example in `Seq`.

```
duration w = 4/4;
Seq seq1 = <1 1 5 5 6 6 5..2/4>;
Seq seq2 = <4 4 3 3 2 2 1..2/4>;
Seq seq3 = <5 5 4 4 3 3 2..2/4>;
Seq seq = [seq1, seq2, seq3, seq3, seq1, seq2[0:len(seq2)-1], 1..w];
```

Additionally the sequences module has built in functions to modify sequences programmatically.

```
Seq phrase11 = {3 2 1 2} Rhythm(beats, {3 3 3});
Seq phrase2 = { phrase11[0:len(phrase11)-1] 3 1};
```

For example, some of these functions allow the user to algorithmically modify and "improvise" new sequences based off of the last notes of a sequence. Other functions allow the addition of notes on two sequences simultaneously by appending notes that follow a specific musical motion

```
Seq::add_note([input_sequence], [semitone/tone], [higher/lower/equal])
Seq::contrary_motion([first_sequence], [second_sequence], [up/down])
```

3.3 Lexical Conventions

3.3.1 Keywords

Below are the keywords `Beathoven` reserved for itself.

```
and    bool    break    char    else
double    duration    false    for    func
if    in    int    pitch    range    return
string    true    unit    while
```

Some of `Beathoven`'s keywords start with upper-case characters. These keywords denote non-primitive data types.

```
Note    Seq    Struct
```

Lastly, one should note that certain`pitch` values such as `C`, `F4#`, `Ab` are also reserved.

3.3.2 Identifiers

Identifiers are character sequences used for naming variables, functions and new data types. Valid characters include ASCII letters, digits, underscores, and an optional apostrophe at the end. A valid name can only start with a letter. So, accepted idenfiers are `(letter) (letter | digit | '_')* '''?`.

Here are some valid identifier examples, `id`, `pitch'`, `beat_hoven`, and `a1_`.

3.3.3 Operators

In Order of decreasing precedence

Precedence	Operators	Description	Associativity
1	..	`Note` constructor, `C4..1/4`	Right
2	.	Membership access operator, `struct.field`	Left
2	[]	Array/Seq access operator	Left
3	!	Not (logical operator)	Right
4	/	`duration` constructor, `1/4`	Nonassoc
6	+ , -	Arithmetic operator	Left
7	!=, >=, ==, <=, >, <	Relational operators	Left
8	and, or	Logical operators	Left
9	:	Index range operator, `arr[1:3]`	Nonassoc
10	=	Assignment	Right

3.3.4 Comments

`Beathoven` allows for either one-line or multi-line comments. Any text after `//` or between `/* */` are ignored. Comments cannot be nested.

```
// Beathoven comment

/* Beathoven
   comments */
```

3.4 Statements

3.4.1 Variables Declaration and Scoping

Declaration

Every name has a type which determines the operations that can be performed on it. A declaration is a statement that introduces a name into the program and specifies its type.

```
Note note = C4;
Seq seq;
pitch[] pitches = [1, 2, 3, 4, 5, 6, 7];
```

Scoping

Variables declared within a function are local and only visible to that function. Variables declared outside functions are global. Local variable that has the same name with some global varible will hide that global variable. ## Control Flow

if/else

Users can write conditionals in `Beathoven` by writing either:

```
if (bool-expression) {
    /* statements */
}
```

Or

```
if (bool-expression) {
    /* statements */
} else {
    /* statements */
}
```

while

The `while` statement executes a block of code while the condition is true. A `break` statement can be used to terminate the execution of the loop.

```
while (condition) {
    /* statements */
}
```

for and range

For loops executes the statement within the braces.

```
int i;
for (i = 0; i < 100; i = i + 1) {
    /* statements */
}
// Here is a syntax sugar for above for loop
for i in range(0, 100) {
    /* statements */
}
```

3.4.2 Functions

Functions are defined using the `func` keyword.

Users can define any kind of function they want by following the format below:

func *Function-identifier* (*arg1-type arg1-identifier . . . argN-type argN-identifier*) -> *return-type* {*statements*}

```
/* A function that returns the unit type, i.e. nothing */
func print_helloworld() -> unit {
    print("hello world");
}


/* A function that returns a Chord */
func Major(pitch base, int inversion) -> Chord {
    if (inversion == 0) {
        return Note(base) & Note(base + 4) & Note(base + 7);
    }
    /* more statements */
}


/* overwrite the duration of 'beats' to 'melody' */
func Rhythm(Seq beats, Seq melody) -> Seq {
    if (len(beats) == 0) {
        print("empty beats");
        return melody;
    }
    else {
        int i = 0;
        int j;
        for j in range(0, len(melody)) {
            melody[j].d = beats[i].d;
            if (i + 1 < len(beats)) i = i + 1;
            else i = 0;
        }
    }
    return melody;
}
```

3.4.3 Function Listings

1. print(...): Accepts variable arguments and multiple data types, and print them.
2. len(Array array): Accepts any array type, and returns the number of array elements.
3. str_of_pitch(pitch pitch'), str_of_duration(duration duration'), str_of_Note(Note): Returns the string of a pitch, a duration or a Note.
4. render_as_midi(Seq seq): Output seq to a Midi file.
5. render_seqs_as_midi(int num, Seq ...): Out sequences to a Midi file (multi-part)

Chapter 4

Project Plan

4.1 Planning Process

4.1.1 OCaml Language

- Intially, we followed the example MicroC to create to create all the functions in MicroC using our language's syntax. We included all those basic operators, so that we decrease the number of reduce/shift errors in parser. After this, we gradually added operators one-by-one which made sense for our language. Just as this additional code was iterative, our testing was iterative. For each additional function, we notified our tester, Eunice, and she implemented a test for that part of the language.
- For example, for semantic checking, at first we only had the AST being generated from the parser, so our code just checked the AST (similarly to MicroC, with duplicate function names).
- At this time, we started working on codegen. At first we had only variable declarations and assignment, but as semantic checking added code we also added code for codegen. In this process, the AST was restructured many times.
- However, after working on print functions, but we realized we needed SAST to print strings as variables. We therefore created the analyzer to produce SAST, and moved all semantic checking to analyzer. This was part of an overall refactoring of our code, as once we needed code files we added them - for example environment and exceptions were moved to separate files. In this process, we added `func_decl`, which declared functions, to our parser, then created pretty print (pprint) to output SAST so easier to debug.
- Last, we started working on structs, which has a lot to do. We have implemented struct assignment, Array, and our own data type note, which is a struct with pointers to two data types (pitch and duration).

4.1.2 C Library Linkage and Architecture

Beathoven uses a C++ Midi Library that it calls internally to render midi files. This section describes how our modules and helper scripts are linked together in order to correctly render these midi files, and how we arrived to our final design. Below are the highlights of our work:

- Recreate MIDI Library - we originally attempted to recreate a MIDI Library in Ocaml with Beathoven. We chose a Python MIDI library that we found as our template. However after talking to professor Edwards, our proposal was deemed to not be complex or "meaty" enough. Professor Edwards suggested we use any C or C++ Library to render midi files instead. Additionally our compiler should compile to llvm code and call the MIDI library with our outputted code to render the MIDI files described in our language's source programs.

- MIDI Library ll modules - After following professor Edwards advice we ran into implementation issues when converting the scripts that used the MIDI Library into ll files. clang is the tool in the llvm program suite that converts c/c++ to ll code. However linking scripts that use external library's using clang's -emit-llvm tool is an experimental use case, as verified by the TA that was assisting us (David Watkins). In brief, clang's -emit-llvm option was trying to link the MIDI library included in our c++ script. Unfortunately, clang cannot link any libraries when doing a c/c++ to llvm conversion with any library other than the standard c/c++ library. We tried several options, like creating the library as a static or a dynamic library and including them when calling clang's -emit-llvm with the -l flag, but clang still produced errors when doing so. We also tried to link the library at a later stage of the command but at that point clang has a system default that exclusively links the file with the standard libraries. LLi cannot run these outputted ll files since it considers the midi library functions as unknown external functions. As a last resort, we attempted to convert all the source files of the MIDI library into ll modules and link them together amongst themselves and with any scripts that use them. However we found that the files are too interdependently linked to attempt such a thing and we still had errors whend trying.

- MIDI Library Wrapper - Given our previous development, we ideated a solution by linking our outputted source code with a wrapper script that compiles a source program in the MIDI library into an executable and calls the executable to render a resulting midi file. This solved our issue since the wrapper script only uses standard c/c++ commands and is thus able to be converted into ll code that our outputted source code could link to. In order for our above solution to be implemented correctly we had to make several changes in our architecture. We currently output a description of the midi sequences described in our source program into a text file hosted in the same directory of the previously mentioned source program we use to render midi. Then our internal library compiles and calls this source program that in turn uses the description of a midi file in the text file to render an actual midi file. Given our development and the alternatives we went through, this option was considered the optimal and thus is how we currently link and render midi files.

4.2 Programming Style Guide

As a team we used a plugin for OCaml formatting and indentation: https://www.typerex.org/ocp-indent.html

As a group we agreed to adhere to the following rules as much as possible:

- No lines greater than 80 characters
- Use tabbed indentation and spaced indentations. Ensure that the tab width is 4 spaces.
- Using meaningful names for all files and functions.
- Indent to indicate scope.
- Wrap lines at 120 characters.
- Comments are not required, but are included for for explanatory purposes, generally before the start of functions.
- Pattern match as much as possible.
- Use a pipe character | with all match cases, including the first one.
- Be as specific as possible when throwing exceptions.
- Do not repeat code — refactor if possible.
- Use as little mutability as possible.
- Be descriptive and consistent in naming everywhere.
- Use lowercase letters and underscores in naming.

4.3 Project Timeline

Date Complete	Goals
November 7th	Test suite Resolve parser ambiguities Start semantic analysis Preliminary codegen for "hello world" - printing, variable assignment, type recognition Discover appropriate C Library for LLVM to MIDI conversion Understand linking for codegen C Library
November 14th	Ast Sast Semantic analysis with continued analyzer work Codegen for llvm built-in functions Test suite for continued functionality Find way to compile LLVM to MIDI
November 21st	Hello World DONE Completed sast Continued work on semantic analysis Continued work on codegen if/while/for statements Continued work on test suite Investigate other ways to compile LLVM to MIDI other than a wrapper
November 28th	Continued work on semantic analysis Continued work on codegen - structs, sequences, Beathoven-specific functions Continued test suite Implement LLVM to MIDI conversion
December 5th	Continue work from last week Implement added Beathoven specific functionality - functions for MIDI conversions, etc
December 12th	Start final report Continue work above until all specifications from LRM are complete
December 19th	Final Report Complete
December 20th	Project Complete

4.4 Team Implementation

4.4.1 Team Roles

- Eunice Kokor - Tester
- Jake Kwon - Language Guru
- Rodrigo Manubens - Musical Guru/System Architect
- Ruonan Xu - Language Guru
- Sona Roy - Manager

4.4.2 Team Responsibilities

Team Member (Github Username)	Responsibility
Eunice Kokor (eunicekokor)	Primary: Test suite Secondary: Analyzer, Ast, Codegen, Environment, PPrint, Semant
Jake Kwon (JakeKwon)	Primary: Analyzer, Semant Secondary: Ast, Environment, Parser, PPrint, Sast, Scanner
Rodrigo Manubens (manubete)	C Library, C Linkage, MIDI conversion
Ruonan Xu (RadonX)	Analyzer, Ast, Codgen, Environment, Sast, Scanner, LLVM to C Linkage, Parser, PPrint
Sona Roy (sonaroy)	Scanner, Parser, Codegen, Analyzer, Sast, Environment, Final Report

4.5 Software Development Environment

We used LLVM 3.8, clang, opam core, and yojson. We also used Github for version control. Each of us used different text editors, but for the most part Sublime was used for code editing.

4.6 Project Log

4.6.1 Beathoven

Detail	**Author**	**Description**
5adf0ec	Jae Hyun Kwon	Initial commit
7683b63	Jake Kwon	testing
b4a9340	Jake Kwon	scanner.mll first draft
70a2a96	RadonX	update scanner.mll
2625d44	RadonX	add .gitignore
bb17198	RadonX	add ast.mli
fc29419	RadonX	create parser.mly
39c92ee	sonaroy	added remaining tokens to parser and added operators with precedence to parser
d569b1f	Jake Kwon	second part of parser
e137ccb	Jae Hyun Kwon	Merge pull request *1 from JakeKwon/sona
4cb478c	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into jake
b5525ec	Jake Kwon	semi to sep
209012a	Jake Kwon	yayay
4534d2f	Jake Kwon	noelse
ef17bd1	Radon Co	Merge pull request *2 from JakeKwon/jake

1a3f500	RadonX	parser expr
92ca3bf	RadonX	merge conflicts
da71305	RadonX	compile script
b2a3630	Eunice Kokor	added preliminary test thing
78b8b1e	Eunice Kokor	Merge branch 'euni-comment'
ec58ba2	RadonX	fix parser.mly
e5182dd	RadonX	fix everything
acc11bc	RadonX	remove duplicate tokens in parser
f74ac00	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into euni-comment
7ae2baf	Eunice Kokor	makefiles and scanner stuf
c8d0d25	Eunice Kokor	tests for scanner work
e6372dc	eunice	Merge pull request *4 from JakeKwon/euni-comment
6aec66f	Eunice Kokor	quick edit
c114e53	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into euni-comment
87cc290	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
766996d	Eunice Kokor	Merge branch 'euni-comment'
ec65f30	Eunice Kokor	script to compile beathoven files
d317e68	Eunice Kokor	pass and fail tests ready for compiler generated output
2408031	Eunice Kokor	updated readme to include test help
126b9bf	Eunice Kokor	Merge branch 'euni-comment'
f371e48	eunice	Merge pull request *5 from JakeKwon/euni-comment
10bd934	RadonX	update .gitignore and src/Makefile
ac4ebb4	RadonX	create semant.ml
596a08b	Jake Kwon	analyzer and makefile
54509ac	Jake Kwon	analyzer testing
38c70fe	Jake Kwon	check not void analyzer
ce6d81c	RadonX	use ocamlbuild in Makefile; update beathoven.ml
16a6695	Jake Kwon	semantic
919af2e	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
1b28156	Eunice Kokor	Merge branch 'master' into semantic
dc61c99	RadonX	codegen draft
86f8128	Jake Kwon	almost
938c5fd	Eunice Kokor	some tests
3998dcf	Eunice Kokor	mergefix
d65dc17	Jake Kwon	finished analyzer (no error)
b0b3a1b	Jake Kwon	merge with origin/semantic
32d3354	Jake Kwon	merge semantic with origin
f547450	eunice	Merge pull request *6 from JakeKwon/semantic
a6bb51a	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
6fe3337	RadonX	codegen that only works with assignment

3b6a90b	RadonX	codegen works only with assignment
3e138ff	RadonX	merge branch codegen
615db0f	RadonX	restructure ast (need to update semant); update codegen.ml
c4da3f6	Jake Kwon	first change to analyzer
4b46238	sonaroy	codegen for binary operations
b715e3c	sonaroy	Merge branch 'sona'
e3523b5	RadonX	codegen that works with func_decl list
17def35	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
5e7ddfc	Jake Kwon	referring dice
64c8e1a	Jake Kwon	merge master
808abba	Jake Kwon	analyzer attempt
2ad32c6	Jake Kwon	merge with codegen
08ec192	Jake Kwon	no breaky
91829e9	RadonX	support print integers
c867d75	Jake Kwon	analyzer fix
ec18a47	Jake Kwon	merge conflict fix
25345f9	Eunice Kokor	trying to fix
be698f3	sonaroy	prelim sast (based on Dice)
d809d5d	sonaroy	prelim environment
98239cd	sonaroy	sast formatted for beathoven based on dice
8d4947e	sonaroy	changed naming format to be modular
5d35f61	sonaroy	sast typos fixed
f101f14	Jake Kwon	analyzer check_stmt
5184a6c	Jake Kwon	merge with master
cc5283b	sonaroy	prelim Dice analyzer methods
b229da6	Eunice Kokor	cleaning up some make files and test ones
fd2bfa7	sonaroy	altered naming structures in analyzer methods from Dice
9874604	sonaroy	fixed some typos
ff5a22c	Jake Kwon	fixing
51331cc	Jake Kwon	Merge branch 'jake-break' of https://github.com/JakeKwon/Beathoven into origin
521fa60	Jake Kwon	fixed analyer error. have sast error
794fb06	Eunice Kokor	almost done
2e4b5e9	Eunice Kokor	some fixes to files
b5bf02c	Jake Kwon	no more error
6089c7d	Eunice Kokor	pass files mostly work. we will add more
a441f5b	eunice	Merge pull request *8 from JakeKwon/jake-break
2124b3f	RadonX	add environment; fix sast
48ce5ac	RadonX	first commit of sast analyzer
1c564b3	RadonX	build sast to expr level
4c64467	RadonX	pretty print sast; sast analyzer still has bugs
e071a8a	RadonX	fix sast analyzer bug (which is not); sast and codegen support vardecl, assign and print; todo: binop

03dad15	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
2212409	Eunice Kokor	tryna get the nonfailed dif to not fail
0eaff2e	Eunice Kokor	merge conflicts
83621aa	RadonX	fix test scripts
a66396c	RadonX	adjust format: module access of sast and ast
bafff5c	RadonX	codegen: print string
3e23419	RadonX	parser: func_decl
56dc228	RadonX	parset: arbitrary order of stmt and func_decl
ceb1859	Jake Kwon	analyzer TODOs
4ed6a60	Jake Kwon	analyzer more todos
c759672	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into analyze
5498bf0	Eunice Kokor	all tests pass
6d2ffda	Eunice Kokor	rm *.diff
3595786	Eunice Kokor	new test all for all functionality
6e79251	Eunice Kokor	small fix to analyzer
d937af1	Jake Kwon	analyzer binop equal|neq
e6e978d	Jake Kwon	analyzer binop and|or
571ceb1	Jake Kwon	analyzer binop Less | Leq | Greater | Geq
79c4747	Jake Kwon	analyzer binop done
97e0fcf	sonaroy	function calls codegen working
843801e	sonaroy	codegen_funccall working, added tests in testall.bt
15275d0	Eunice Kokor	if statement analyzer work
e904338	Eunice Kokor	excpetion conflict
573dd8a	sonaroy	fixed merge conflict
2ef3c8e	sonaroy	Merge pull request *10 from JakeKwon/sona
8d7c6eb	RadonX	update README.md
3e3458c	RadonX	support one line comment
083208c	Eunice Kokor	func test initial without parameters
b11a64c	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
f9bf434	Jake Kwon	analyzer checks done
a27e98c	Jake Kwon	minor change
82e6c6d	Jake Kwon	merge Merge branch 'master' of https://github.com/JakeKwon/Beathoven
fa6d777	Jake Kwon	while tests
2594886	Jake Kwon	analyzer unary op and testings
9294e60	RadonX	front end for type pitch
e2eb21f	RadonX	binary operators
b382f22	RadonX	Merge branch "ruonan"
33c4c65	Radon Co	Update README
641209d	RadonX	some llvm linker, structs
665ae74	RadonX	pprint sast, formals
b1f8633	Eunice Kokor	unary operators working in codegen
34f3df0	RadonX	support passing function parameters

83d829e	RadonX	merge codegen
f4d0254	RadonX	fix test (add lib path); remove temporary test file
a8eb219	Eunice Kokor	while codegen complete, test not pass
1b9c410	Jake Kwon	to pull
b5a2839	Jake Kwon	merge
c81f3ed	Jake Kwon	analyzer check_fbody
d8ed777	Jake Kwon	todo:check t is Unit
ae823d0	sonaroy	compiling codegen_if_stmt
78ed5fc	sonaroy	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into sona
a5c6a43	Jake Kwon	getIDtype
e67676a	sonaroy	Merge pull request *11 from JakeKwon/sona
42b355f	Eunice Kokor	for attempt
043ed25	Jake Kwon	check duplicate and reserved func
6008b43	Eunice Kokor	it's failing again
0fe016d	Jake Kwon	duplicate test works
feb0b0c	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
b57c8bb	Jake Kwon	Merge branch 'analyzerTodos'
0ac8b18	Rodrigo	added bet_midi_library as a git submodule
b082ac5	Eunice Kokor	not breaking everything
8154e5c	sonaroy	added if/else testing
6643e77	sonaroy	Merge pull request *13 from JakeKwon/sona
3829f64	sonaroy	Merge pull request *12 from JakeKwon/rigo_parser
debe59d	Eunice Kokor	fixmerge
8ccfe57	Eunice Kokor	forgot to remove merge conflict
d3612c3	Eunice Kokor	no errs
5037602	RadonX	if structs are stmt
4d38ecc	RadonX	let struct only be defined in main func
8113036	RadonX	extract structs to sast from main func for every module; simple parser / complicated analyzer trade-off
1a67b4c	RadonX	merge branch "ruonan" (frontend of structs); improve checking builtin funcs
f91a783	RadonX	codegen_struct
d91e3b4	RadonX	Sast struct field
677e739	RadonX	codegen for initialize struct variable and load struct field
e1260ca	RadonX	assignment of struct
11bd11f	RadonX	pitch declaration and pitch literal
894d78f	RadonX	mute stdlib.c so that it won't fail tests; codegen_assign
4f14e9c	RadonX	clean logs directory
87ca5c0	RadonX	Struct assignment, which needs memcpy, which needs sizeof
7b9bc5f	RadonX	Assignment with pitch literal

e1d3758	RadonX	Add author header; Pprint format; Start working on array type
59095a6	RadonX	front end for array decl and assign (nested array will be flattened
5c89f85	RadonX	introduce global variable
a76278b	RadonX	codegen: declare arrays and create simple array literals
1b5d674	RadonX	refactor load_id, codegen_expr_ref; create Arr struct
1242f25	RadonX	front end for ArrayIdx and ArraySub
41c421b	RadonX	Array Access with idx
4e0e5bf	RadonX	Access field of struct array element, like arr[0].field
19e3d22	RadonX	Rename Datatype as Primitive
3881b01	RadonX	refactor analyzer.ml: binop
2154b2f	Eunice Kokor	fix merge
18ae98b	RadonX	fix empty array issues; vardecl type check
a79773a	RadonX	print pitch with stdlib.c; pitch.key waiting to be supported (char type)
c22cabf	Eunice Kokor	fail tests working, globallog for fail could maybe use work
d81d66d	Eunice Kokor	fail tests working, globallog for fail could maybe use work
9f5d34c	eunice	Merge branch 'master' into test-fail
8890f12	eunice	Merge pull request *15 from JakeKwon/test-fail
e4db6e2	Eunice Kokor	fix merge conflict
56a7d39	Eunice Kokor	uncomment out linker
779c974	RadonX	front end for duration type and literals; refactor: sast, environment
7d30dbc	RadonX	fix merge conflicts
31818d9	RadonX	add failure tests output
da5949f	RadonX	global vardecl in main funcs of modules
a98fdeb	RadonX	update analyzer and environment to support global vardecl in codegen
fada50f	RadonX	utils for initialize beathoven primitive types (struct in llvm)
9ea91ff	Eunice Kokor	I think merge conflicts are gone
1f51707	Eunice Kokor	yay
9ea269e	eunice	Merge pull request *16 from JakeKwon/whilecodegen
49f284d	sonaroy	added char and author comments
b8fea09	sonaroy	Merge pull request *17 from JakeKwon/sona
13084cf	RadonX	duration literals in codegen; _str_of_duration in stdlib.c
e9346c0	Jake Kwon	struct pass/fail tests
2ab45d7	Jake Kwon	solve merge conflict
b5119ee	Jake Kwon	solve merge conflict
2570e93	Jae Hyun Kwon	Merge pull request *18 from JakeKwon/jakeTesting
aaae6d0	Jake Kwon	tested weird case ruonan brought up
d30487c	Eunice Kokor	yay fail and pass tests separate

d2ee5f5	Eunice Kokor	globallog conflict
e2d0c08	Eunice Kokor	compiler exception testing done
11bd610	RadonX	fix char parsing error (in ast)
0b1765a	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
9cea026	RadonX	mute debug
7a69fa3	RadonX	fix an error of building main func twice
7073dc5	RadonX	merge codegen: duration literals
6f3a8d2	Eunice Kokor	test
7a56afd	eunice	Update .travis.yml
fa1cd3c	eunice	Update install.sh
0c5d068	eunice	Update .travis.yml
9a9c0c6	eunice	Update install.sh
6f3e846	Jake Kwon	prettier parser error
8da4110	Jake Kwon	prettier parser error test
41c3853	Jake Kwon	solve merge conflict
62ce7ac	Jake Kwon	merge conflict
352ad68	eunice	Update .travis.yml
196a753	Rodrigo	added functions to stdlib.c
f5e192a	eunice	Update .travis.yml
a104860	eunice	Update .travis.yml
5318670	eunice	Update .travis.yml
34a1fb2	eunice	Update .travis.yml
5f128b2	eunice	Update .travis.yml
3567fb9	eunice	Delete .travis.yml
b289d4a	RadonX	beathoven.h for essential types in our language
8c43dbb	RadonX	remove log file
67022c3	RadonX	but keep logs folder
1bbfcb2	Eunice Kokor	added the rest of testall.bt tests
39434b3	eunice	Merge pull request *22 from JakeKwon/exception-testing
23d23cb	Jake Kwon	struct array
fae495d	Jake Kwon	merge conflict
b215248	RadonX	using easier beathoven types
7b8c149	eunice	Create .travis.yml
b669568	eunice	Update .gitignore
0fc9073	RadonX	most of the types in beathoven.h; codegen: pitch -> _pitch*; pitch and duration have bugs in assignment
b7602dc	RadonX	log.ml to help debug; fix duration assign error (load_id)
a3288a4	RadonX	change pitch and duration as Primitive() types
68ec6ce	RadonX	Merge branch 'codegen' Change pitch and duration as Primitive() types; add log.ml for debug.
2da9269	Rodrigo	merged master
75d8708	Rodrigo	merged the remote changes on rigo parser
581f00c	Jake Kwon	pitch/duration testings

9649b88	RadonX	finally make COLON work in both Note literals and subarray index
fe02f7d	Jake Kwon	declaration/assignment for all types
ba7bf86	Jake Kwon	type check for paramaters
6b73356	RadonX	other front end stuff for Note; a little codegen refactor
20a09b4	Jake Kwon	analyzer param type check and testing
1895b09	RadonX	codegen: Note
a0767fa	RadonX	Merge branch 'ruonan': implement Note; change Parser a lot.
982ce1f	Jake Kwon	trying some arith tests
02c28c3	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
05b15f4	Eunice Kokor	codegen return
363b050	eunice	Merge pull request *24 from JakeKwon/whilecodegen
d148b48	Jake Kwon	let pitch take an integer
946ce38	Jake Kwon	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
7524f9a	Eunice Kokor	while
d067f66	Eunice Kokor	while progress
deb97d0	Rodrigo	tested methods that we will incorporate as functions in the library
9a2cd6e	RadonX	prepare parser for include list; create stdlib.cpp and stdlib.bt
e8e096e	RadonX	Merge branch 'ruonan'
bebd7c8	Eunice Kokor	progress
e0c251d	Eunice Kokor	file output
676c458	RadonX	codegen_while is good; but something is wrong with main ret void
d842fe2	RadonX	fix a builder bug
23c1865	Rodrigo	Added the lli modules to write notes to the destination midi text file, and render midi from them
c45e66d	Rodrigo	merged master
81df50b	Rodrigo	commented out stub code
5b471a4	Eunice Kokor	hahah
ee359fe	RadonX	finally make some sound
c759e65	RadonX	finally make some sound
09e0247	Eunice Kokor	file I/O
4b85a71	Eunice Kokor	working with file input
bf72120	eunice	Merge pull request *25 from JakeKwon/whilecodegen
8ef9e97	Eunice Kokor	merge conflict fix
a88eb9c	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
2aa09df	RadonX	Front end for Seq; not any conflicts in parser!!
5d4afd2	RadonX	Merge branch 'ruonan' into codegen
4ba8b28	Eunice Kokor	yay
5f4365b	eunice	Merge pull request *26 from JakeKwon/whileTest
d96bc43	RadonX	codegen for Seq; Note works with LitInt

58c29ad	RadonX	merge codegen: Seq
26ee5bb	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
4fc4fc4	RadonX	'for' statement; remove a lot of unused stuff in analyzer.ml
7d662af	RadonX	Merge branch 'codegen'
379cec9	RadonX	parser is clean again when introducing a new token
8514582	RadonX	parser: full support for notes in Seq
da971af	RadonX	Analyzer: cast Int, Pitch to Note type
7a89198	RadonX	builtin len() for all types of array
23e44c0	Eunice Kokor	git push origin whiletest
1633a67	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into whileTest
bf65990	RadonX	working on arraysub
3de226f	Eunice Kokor	tests not passing but important
15590d7	RadonX	memcpy subarray in codegen done. headache now.
0d3d890	Eunice Kokor	really testing music codegen
e2e442a	RadonX	front end for array concat
810a73e	RadonX	concatenate array in LLVM shit
8028cd3	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
8304d5c	RadonX	fix some music tests
6918517	RadonX	Seq operations
de3a9ce	RadonX	fix cast error
637f0f5	RadonX	example code: twinkle
78090e7	RadonX	add stdlib.bt; cast duration in LitSeq; add 'for in range'
0494fe8	RadonX	Fix if statement; Implement break statement
7e389d4	Eunice Kokor	clean test option
b3b5beb	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven into clean-testout
87f01cd	Rodrigo	local changes
878afdc	Rodrigo	merged master
8449505	RadonX	Change Note as structtype; add example/rhythm.bt
7a77937	Eunice Kokor	cleaner test
b9a99ef	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
53918f7	Eunice Kokor	Merge branch 'clean-testout'
00a85ab	Eunice Kokor	failtestfix
56be93e	RadonX	Support multiple seqs midi
b9d4b4b	RadonX	Merge branch 'ruonan'
86ce8ae	Rodrigo	Merge branch 'master' of https://github.com/JakeKwon/Beathoven Merging master to get new methods
d486ac8	RadonX	1ând 7_; default index

5d1d0d0	Eunice Kokor	example script
bc88544	Rodrigo	Added new examples
31060b2	Eunice Kokor	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
b4c0e38	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven
9d8e8bc	RadonX	Merge branch 'master' of https://github.com/JakeKwon/Beathoven

4.6.2 Midi Library

Detail	Author	Description
3ed8a31	Rodrigo	modified and renamed the main source program, made cpp scripts and ll modules for use in the beathoven language
e92f8ec	Rodrigo	modified README.md
da81470	Rodrigo	added changes to bet midi library so it works with doubles in note and duration intake

Chapter 5

Architecture

5.1 Diagrams

5.1.1 Compiler

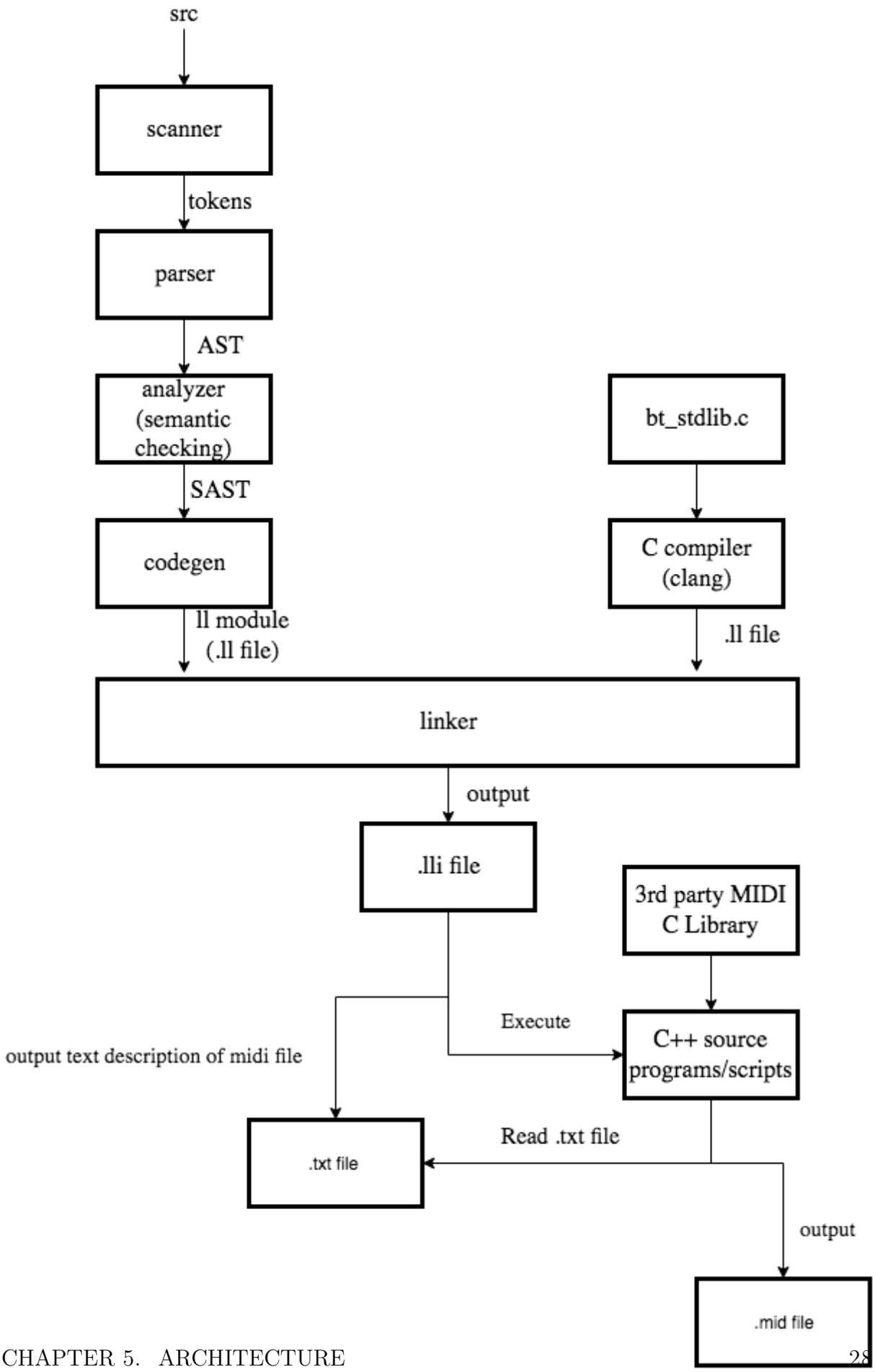

Figure 5.1: The Compiler Architecture for Beathoven

1. The source code is input into the scanner, which generates tokens. These tokens are the input into the parser, which checks for syntax by generating the abstract syntax tree. This AST is then fed into the analyzer, which implements semantic checking and outputs the semantically checked abstract syntax tree. This SAST is then input into codegen, which generated the intermediate code representation in an ll module (a module using LLVM), which creates a .ll file.

2. We have a third party MIDI library, which outputs the necessary C++ source programs and scripts for C++ to MIDI conversion. This is compiled using clang, the C language family frontend for LLVM, into a .ll file.

3. The standard library for our language is created in C, then compiled using clang to output a .ll file.

4. Each of these .ll files are linked, and the combination of all three steps generates an output MIDI file.

5.1.2 AST

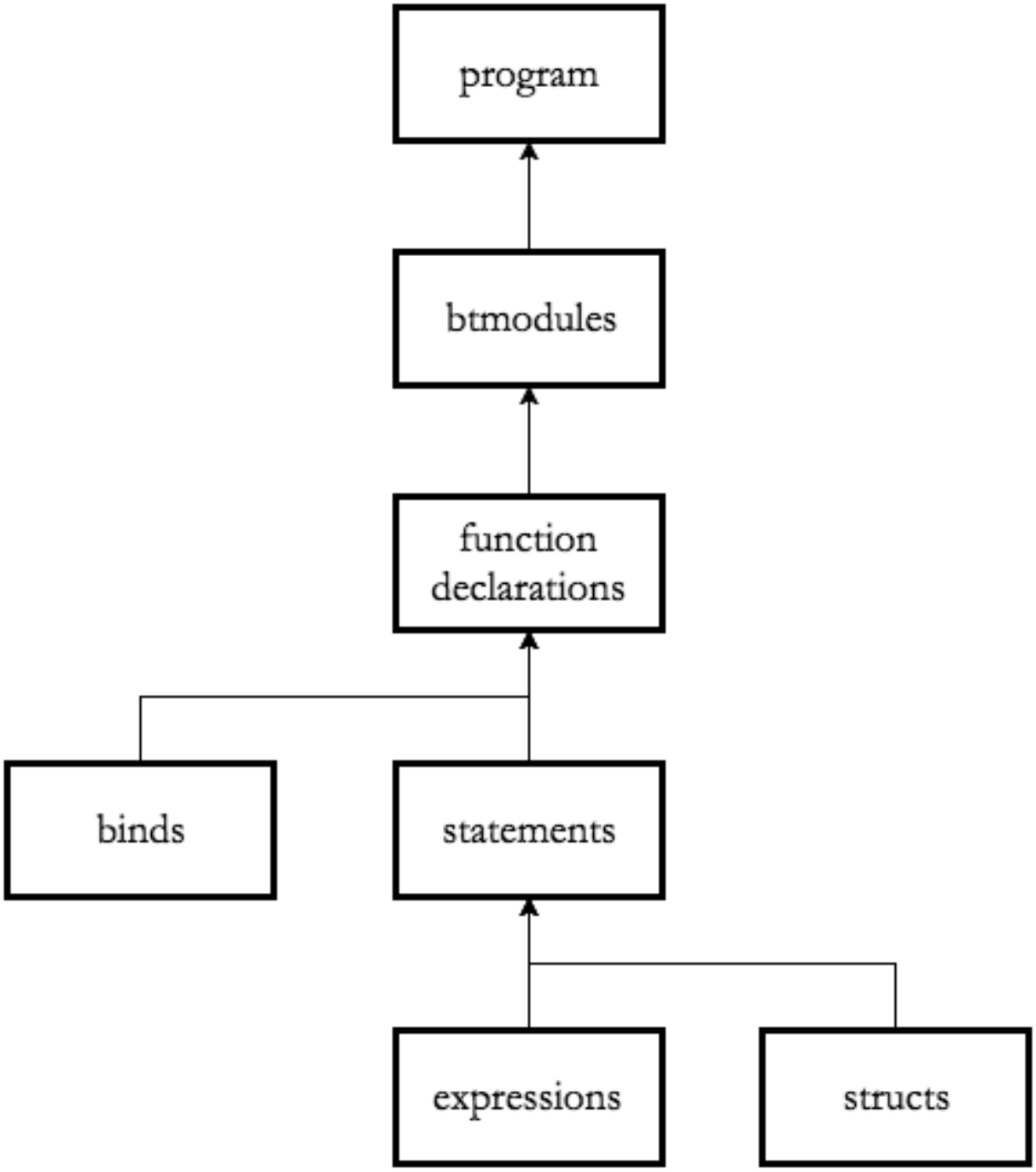

Figure 5.2: Beathoven's Abstract Syntax Tree

5.1.3 SAST

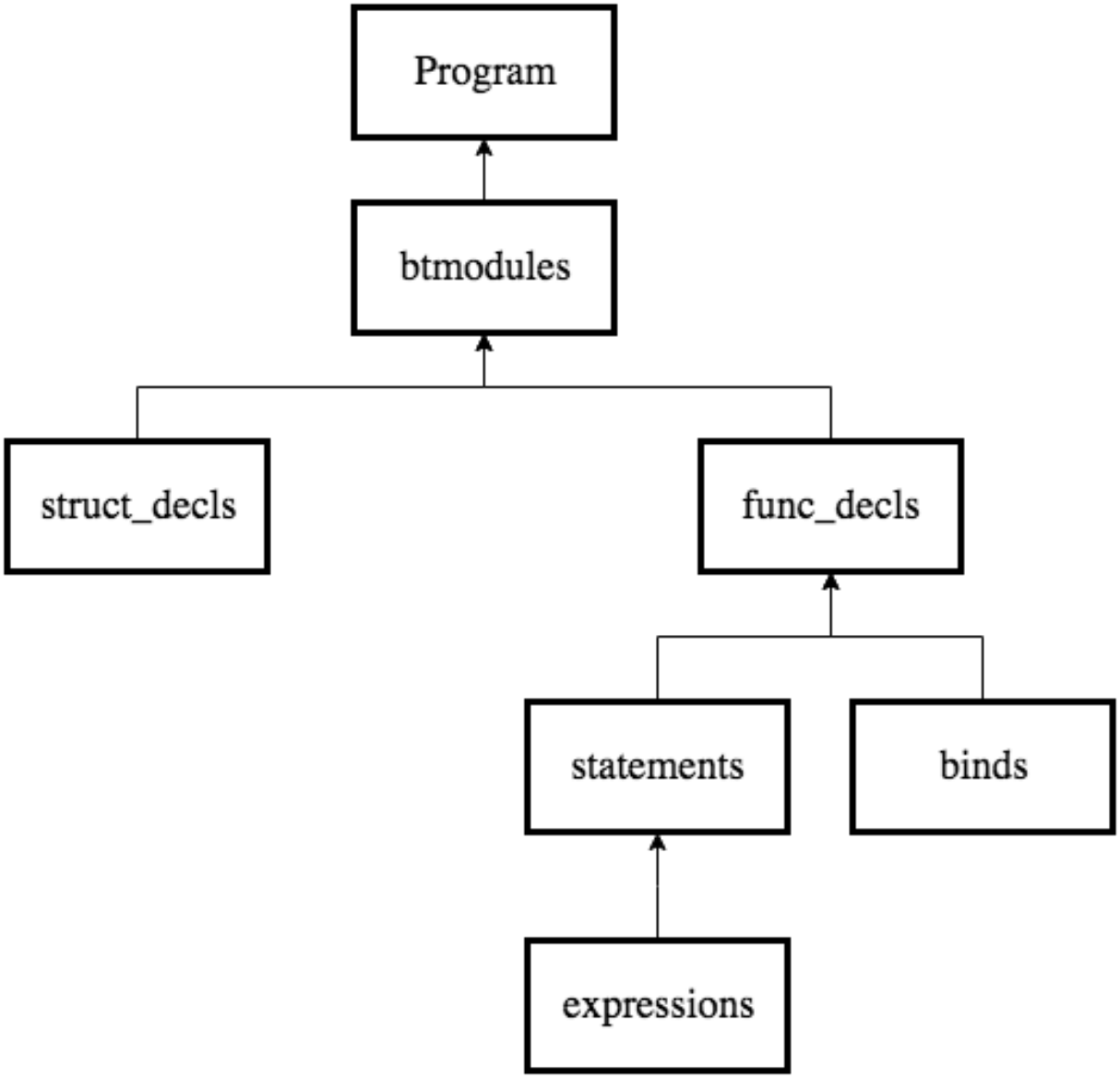

Figure 5.3: Beathoven's Semantically-checked Abstract Syntax Tree

5.2 Interfaces Between Components

1. `Makefile` - our makefile, to generate compile commands dependent on flags given in terminal
2. `analyzer.ml` - Semantically checks incoming AST representation to make sure that it includes existing files, adheres to the rules of inheritance, and expressions are properly type-checked
3. `ast.ml` - our abstract syntax tree, generated from the parser to give contextual analaysis
4. `beathoven.ml` - main file of our compiler program

5. `codegen.ml` - our code generation, where we coverted our code into an intermediate representation of LLVM
6. `environment.ml` - allows us to see global variables within functions
7. `exceptions.ml` - our file of all exceptions in our language
8. `parser.mly` - Reads in tokens from the scanner to produce an AST representation of the program
9. `pprint.ml` - pretty print, allowing us to print the SAST in a pretty manner for debugging purposes
10. `sast.ml` - semantically checked abstract syntax tree, used for infer types from variables
11. `scanner.mll` - Reads a source file and tokenizes it to the corresponding token output
12. `stdlib.c` - the C standard library, used for codegen and linkages
13. `stdlib.ll` - the LLVM standard library, used for codegen and linkages
14. `testall.bt` - all the implemented features of the language, updated when content added to `codegen.ml`

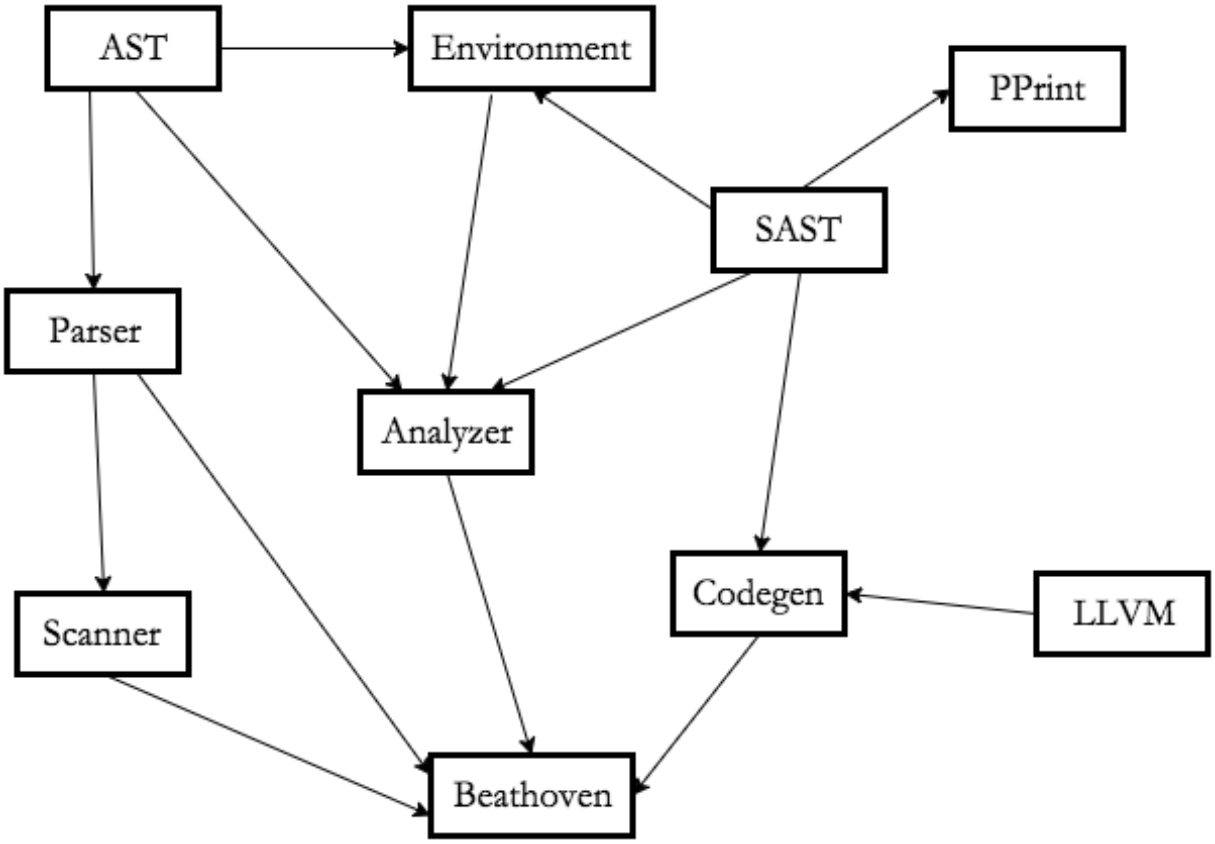

Figure 5.4: File Dependency Graph. The arrows pointing out indicate which other components of the compiler that particular file/aspect is dependent upon.

Chapter 6

Test Plan

6.1 Testing Phases

6.1.1 Unit Testing

Scanner, parser, codegen was tested as the semester went on. At the beginning stages, we often tested input `.bt` files and fed it into a scanner to see if the correct tokens were being matched. We test with various input strigns the different cases of input by injecting those cases in to the scanner and printing them. See the example of the code below. The parser was tested by printing a semantically checked JSON Tree of our results. See sectin 6.3.2 for an example of this. Code generation was hand tested with sample programs in our integration testing.

Beathoven File Input	Tokenized Output
int a = 1;	TYPE_INT ID EQ LIT_INT SEP

6.1.2 Integration Testing

Testing syntax and expressions to sanity check the compiler with our language end-to-end from scanning tokens to code generation. The generated code is then compiled with `lli` and run to verify the expected output and/or result.

- Keywords: Keywords are a subset of identifiers which are reserved for our language. To verify, sufficient testing was conducted on cases such as int a; for a positive case or int if; for a negative case.
- Control flow: Control flow was unit tested in the following way: if, if/else, while, and return. Negative cases were also tested, such as having an else by itself.
- Types: We support 5 primitives types int, char, double, bool, and string. We tested those types with negative test cases such as assigning an int to a string. We have chars implemented for pitch.

- Arrays: We test the declaration and usage of arrays of different types to verify the correctness. We use printing to verify the result. We also tested the various pass by methods.
- Struct: We tested the fact that structs were pass by value, by copying one struct to another and printing specific values we wanted.
- Built-in functions: We tested music printing str_of_pitch, str_of_duration, str_of_Note, render_as_midi, render_seqs_as_midi
- Comments: We used comments scattered in our code generously to test multiline and inline comments.
- Operators Binary and unary operators were tested by using them and checking their correctness after applying the operator.
- Variable and function declarations Here we implement global and nested variables and functions. We try different parameters for functions and different return types.

6.1.3 System Testing

This testing phase was to answer if beathoven not only generated code-generated outputs, but also midi outputs. You will see examples of this in our sample programming. There wasn't a good way to test input/output of this, but rather if the `lli .bt` resulted in a new midi file in the `bet_midi_library` folder or not and playing the sample file on `https://onlinesequencer.net/`. Most of our real system tests are located at `examples` folder.

6.2 Automation of Testing

Automation Our compile script, beathoven.sh, in the main beathoven directory takes in a all the files within our compiler pass and compiler fail directories (must have extension ".bt") in that directory to LLVM code that can be compiled with lli and run. Then the outputs of running that llvm code is compared to an expected output (.err for fail tests and .out for pass tests).

6.2.1 Test-Related Automation Commands

- `make compile-test` compiles and links then runs `all_test.sh` test script or `make tests` without compiling and linking
- `make compile-pass` compiles and links then runs `pass_test.sh test script` or `make tests-pass` without compiling and linking
- `make compile-fail` compiles and links then runs `pass_test.sh test script` or `make tests-fail` without compiling and linking

6.3 Test suites

There are two folders with our tests. Compiler pass with test cases that should pass and compiler fail with test cases that should fail. As specific features were implemented, the beathoven code was added to `/src/.testall.bt`, then we created small integration tests to test the functionality of what was just implemented. We also created fail tests for all the possible exceptions that could be raised in our exeptions.ml file. These are manually run using the `beathoven` binary produced by the make command in the top level directory.

6.3.1 Test Output

We wanted our tests to be as user friendly as possible. We have an option in our test scripts called `helperPrint` which can be set to true, there can be printed out the commands that were being automated, so that bugs can be reprouced to see exactly what went wrong. We also kept a `logs/` folder to display the intermediate .ll, final `.out` output, and `.diff` outputs of all tests as well as a folder `diffs` for all the failed tests. For every test, the entire standard output and standard error was piped to a `logs/globallog.log` file so we could track over time which tests were passing or failing with each commit. If a test fails it prints "FAILURE REASON. See globallog.log file for breakdown". We have provided the full output for the first tests in the suite, then we shortened the rest for brevity.

Listing 6.1: Sample Test Run

```
$ cd compiler; ./pass_test.sh
##### Running Compiler pass Tests! #####


Running Pass Test: all      SUCCESS
Running Pass Test: arithmetic       SUCCESS
Running Pass Test: array      SUCCESS
Running Pass Test: array_pythonic     SUCCESS
Running Pass Test: bool       SUCCESS
Running Pass Test: break      SUCCESS
Running Pass Test: char       SUCCESS
Running Pass Test: double     SUCCESS
Running Pass Test: duration       SUCCESS
Running Pass Test: functions      SUCCESS
Running Pass Test: helloworld       SUCCESS
Running Pass Test: if       SUCCESS
Running Pass Test: loopy      SUCCESS
Running Pass Test: math       SUCCESS
Running Pass Test: music      SUCCESS
Running Pass Test: music_to_string    SUCCESS
Running Pass Test: note       SUCCESS
Running Pass Test: pitch      SUCCESS
Running Pass Test: printStr       SUCCESS
Running Pass Test: sequence       SUCCESS
```

```
Running Pass Test: string      SUCCESS
Running Pass Test: stringequation      SUCCESS
Running Pass Test: struct      SUCCESS
Running Pass Test: structarray      SUCCESS
Running Pass Test: while      SUCCESS
You have 0 out of 25 PASS errors

#### End of Pass Compiler Tests! ####
cd test; make tests-fail
cd compiler; ./fail_test.sh

#### Starting Fail Compiler Tests! ####
Running Fail Test: arrtypenotmatch  SUCCESS
Running Fail Test: duplicate-func  SUCCESS
Running Fail Test: duplicate-vardecl  SUCCESS
Running Fail Test: elseonly  SUCCESS
Running Fail Test: func_not_found  SUCCESS
Running Fail Test: keyword_fail  SUCCESS
Running Fail Test: param_type_not_match  SUCCESS
Running Fail Test: parser_err  SUCCESS
Running Fail Test: shouldaccessstructtype  SUCCESS
Running Fail Test: structfieldnotfound  SUCCESS
Running Fail Test: type_addition  SUCCESS
Running Fail Test: typemismatch  SUCCESS
Running Fail Test: undefinedstruct  SUCCESS
Running Fail Test: weird  SUCCESS
You have 0 out of 14 FAIL errors

#### End of Fail Compiler Tests! ####
```

6.3.2 Pretty Printing

Listing 6.2: pitch.bt input file

```
/* pitch */
pitch p = C4;
pitch q;
q = C4;
pitch r = 2;
string r_string = str_of_pitch(r);
print("This pitch is ", r_string);
```

Listing 6.3: Sast tree output

```
$ ./beathoven.sh -s < ../test/compiler/pass/pitch.bt
{
  "program": {
```

```
"btmodules": [
  {
    "btmodule": {
      "mname": "__bt",
      "structs": [
        {
          "struct_decl": {
            "sname": "__pitch",
            "fields": [
              { "name": "key", "datatype": "char" },
              { "name": "octave", "datatype": "int" },
              { "name": "alter", "datatype": "int" }
            ]
          }
        },
        {
          "struct_decl": {
            "sname": "__duration",
            "fields": [
              { "name": "a", "datatype": "int" },
              { "name": "b", "datatype": "int" }
            ]
          }
        },
        {
          "struct_decl": {
            "sname": "Note",
            "fields": [
              { "name": "p", "datatype": "pitch" },
              { "name": "d", "datatype": "duration" }
            ]
          }
        }
      ],
      "funcs": [
        {
          "func_decl": {
            "fname": "main",
            "returnType": "unit",
            "formals": [],
            "body": [
              {
                "vardecl": {
                  "datatype": "pitch",
                  "name": "__bt.p",
                  "val": { "pitch": "C4__0" }
                }
              },
              {
```

```
            "vardecl": {
              "datatype": "pitch",
              "name": "__bt.q",
              "val": "noexpr"
            }
          },
          {
            "stmt_expr": {
              "expr": {
                "assign": {
                  "lhs": {
                    "id": { "name": "__bt.q", "datatype": "pitch" }
                  },
                  "rhs": { "pitch": "C4__0" },
                  "datatype": "pitch"
                }
              },
              "datatype": "pitch"
            }
          },
          {
            "vardecl": {
              "datatype": "pitch",
              "name": "__bt.r",
              "val": { "pitch": "D4__0" }
            }
          }
        ]
      }
    }
```

6.4 Beathoven to LLVM (Source to Target)

Listing 6.4: Sample music generation

```
duration w = 4/4;
Seq seq1 = <1 1 5 5 6 6 5..2/4>;
Seq seq2 = <4 4 3 3 2 2 1..2/4>;
Seq seq3 = <5 5 4 4 3 3 2..2/4>;
Seq seq = [seq1, seq2, seq3, seq3, seq1, seq2[0:len(seq2)-1], 1.. w];
render_as_midi(seq);
```

Listing 6.5: LLVM file for music

```
; ModuleID = 'Beathoven Codegen'
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
```

```
target triple = "x86_64-apple-macosx10.12.0"

%_duration = type { i32, i32 }
%Arr_Struct_Note = type { i64, %Note* }
%Note = type { %_pitch*, %_duration* }
%_pitch = type { i8, i32, i32 }
%struct.___sFILE = type { i8*, i32, i32, i16, i16, %struct.___sbuf, i32, i8*, i32
    (i8*)*, i32 (i8*, i8*, i32)*, i64 (i8*, i64, i32)*, i32 (i8*, i8*, i32)*,
    %struct.___sbuf, %struct.___sFILEX*, i32, [3 x i8], [1 x i8], %struct.___sbuf, i32,
    i64 }
%struct.___sFILEX = type opaque
%struct.___sbuf = type { i8*, i32 }
%struct.___va_list_tag = type { i32, i32, i8*, i8* }

@_bt.w = global %_duration* null
@"1/1" = global %_duration zeroinitializer
@_bt.seq1 = global %Arr_Struct_Note zeroinitializer
@C4_0 = global %_pitch zeroinitializer
@"1/4" = global %_duration zeroinitializer
@.litNote = global %Note zeroinitializer
@.litNote.1 = global %Note zeroinitializer
@G4_0 = global %_pitch zeroinitializer
@.litNote.2 = global %Note zeroinitializer
@.litNote.3 = global %Note zeroinitializer
@A4_0 = global %_pitch zeroinitializer
@.litNote.4 = global %Note zeroinitializer
@.litNote.5 = global %Note zeroinitializer
@"1/2" = global %_duration zeroinitializer
@.litNote.6 = global %Note zeroinitializer
@.litarr_Struct_Note = global %Arr_Struct_Note zeroinitializer
@_bt.seq2 = global %Arr_Struct_Note zeroinitializer
@F4_0 = global %_pitch zeroinitializer
@.litNote.7 = global %Note zeroinitializer
@.litNote.8 = global %Note zeroinitializer
@E4_0 = global %_pitch zeroinitializer
@.litNote.9 = global %Note zeroinitializer
@.litNote.10 = global %Note zeroinitializer
@D4_0 = global %_pitch zeroinitializer
@.litNote.11 = global %Note zeroinitializer
@.litNote.12 = global %Note zeroinitializer
@.litNote.13 = global %Note zeroinitializer
@.litarr_Struct_Note.14 = global %Arr_Struct_Note zeroinitializer
@_bt.seq3 = global %Arr_Struct_Note zeroinitializer
@.litNote.15 = global %Note zeroinitializer
@.litNote.16 = global %Note zeroinitializer
@.litNote.17 = global %Note zeroinitializer
@.litNote.18 = global %Note zeroinitializer
@.litNote.19 = global %Note zeroinitializer
@.litNote.20 = global %Note zeroinitializer
@.litNote.21 = global %Note zeroinitializer
@.litarr_Struct_Note.22 = global %Arr_Struct_Note zeroinitializer
@_bt.seq = global %Arr_Struct_Note zeroinitializer
```

```
@.arrsub = global %Arr_Struct_Note zeroinitializer
@.arrsub.23 = global %Arr_Struct_Note zeroinitializer
@.litNote.24 = global %Note zeroinitializer
@.litarr_Struct_Note.25 = global %Arr_Struct_Note zeroinitializer
@.arrconcat = global %Arr_Struct_Note zeroinitializer
@_pitch_values = global [7 x i32] [i32 0, i32 2, i32 4, i32 5, i32 7, i32 9, i32 11],
    align 16
@_buffer = common global [20 x i8] zeroinitializer , align 16
@.str = private unnamed_addr constant [7 x i8] c"%c%d%c\00", align 1
@.str.1 = private unnamed_addr constant [6 x i8] c"%d/%d\00", align 1
@.str.2 = private unnamed_addr constant [12 x i8] c"%c%d#:%d/%d\00", align 1
@.str.3 = private unnamed_addr constant [11 x i8] c"%c%d:%d/%d\00", align 1
@___stdoutp = external global %struct.___sFILE*, align 8
@.str.4 = private unnamed_addr constant [25 x i8] c"Current working dir: %s\0A\00",
    align 1
@.str.5 = private unnamed_addr constant [38 x i8]
    c"/../bet_midi_library/file_example.txt\00", align 1
@.str.6 = private unnamed_addr constant [15 x i8] c"getcwd() error\00", align 1
@.str.7 = private unnamed_addr constant [2 x i8] c"w\00", align 1
@.str.8 = private unnamed_addr constant [2 x i8] c"a\00", align 1
@.str.9 = private unnamed_addr constant [9 x i8] c"Error! \0A\00", align 1
@.str.10 = private unnamed_addr constant [4 x i8] c"%d,\00", align 1
@.str.11 = private unnamed_addr constant [4 x i8] c"-1\0A\00", align 1
@.str.12 = private unnamed_addr constant [4 x i8] c"%f,\00", align 1
@.str.13 = private unnamed_addr constant [13 x i8] c"./betmidi.sh\00", align 1

define void @main() {
entry:
  store i32 1, i32* getelementptr inbounds (%_duration, %_duration* @"1/1", i32 0,
      i32 0)
  store i32 1, i32* getelementptr inbounds (%_duration, %_duration* @"1/1", i32 0,
      i32 1)
  store %_duration* @"1/1", %_duration** @_bt.w
  %0 = trunc i64 7 to i32
  %mallocsize = mul i32 %0, trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
      i1** null, i32 1) to i64), i64 2) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %.arr = bitcast i8* %malloccall to %Note*
  %.idx = getelementptr %Note, %Note* %.arr, i32 0
  store i8 67, i8* getelementptr inbounds (%_pitch, %_pitch* @C4_0, i32 0, i32 0)
  store i32 4, i32* getelementptr inbounds (%_pitch, %_pitch* @C4_0, i32 0, i32 1)
  store i32 0, i32* getelementptr inbounds (%_pitch, %_pitch* @C4_0, i32 0, i32 2)
  store i32 1, i32* getelementptr inbounds (%_duration, %_duration* @"1/4", i32 0,
      i32 0)
  store i32 4, i32* getelementptr inbounds (%_duration, %_duration* @"1/4", i32 0,
      i32 1)
  store %_pitch* @C4_0, %_pitch** getelementptr inbounds (%Note, %Note*
      @.litNote, i32 0, i32 0)
  store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
      @.litNote, i32 0, i32 1)
  %lhs_p = bitcast %Note* %.idx to i8*
  call void @memcpy(i8* %lhs_p, i8* bitcast (%Note* @.litNote to i8*), i64 mul nuw
```

```
    (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx1 = getelementptr %Note, %Note* %.arr, i32 1
store %__pitch* @C4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.1, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.1, i32 0, i32 1)
%lhs_p2 = bitcast %Note* %.idx1 to i8*
call void @memcpy(i8* %lhs_p2, i8* bitcast (%Note* @.litNote.1 to i8*), i64 mul nuw
    (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx3 = getelementptr %Note, %Note* %.arr, i32 2
store i8 71, i8* getelementptr inbounds (%__pitch, %__pitch* @G4__0, i32 0, i32 0)
store i32 4, i32* getelementptr inbounds (%__pitch, %__pitch* @G4__0, i32 0, i32 1)
store i32 0, i32* getelementptr inbounds (%__pitch, %__pitch* @G4__0, i32 0, i32 2)
store %__pitch* @G4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.2, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.2, i32 0, i32 1)
%lhs_p4 = bitcast %Note* %.idx3 to i8*
call void @memcpy(i8* %lhs_p4, i8* bitcast (%Note* @.litNote.2 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx5 = getelementptr %Note, %Note* %.arr, i32 3
store %__pitch* @G4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.3, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.3, i32 0, i32 1)
%lhs_p6 = bitcast %Note* %.idx5 to i8*
call void @memcpy(i8* %lhs_p6, i8* bitcast (%Note* @.litNote.3 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx7 = getelementptr %Note, %Note* %.arr, i32 4
store i8 65, i8* getelementptr inbounds (%__pitch, %__pitch* @A4__0, i32 0, i32 0)
store i32 4, i32* getelementptr inbounds (%__pitch, %__pitch* @A4__0, i32 0, i32 1)
store i32 0, i32* getelementptr inbounds (%__pitch, %__pitch* @A4__0, i32 0, i32 2)
store %__pitch* @A4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.4, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.4, i32 0, i32 1)
%lhs_p8 = bitcast %Note* %.idx7 to i8*
call void @memcpy(i8* %lhs_p8, i8* bitcast (%Note* @.litNote.4 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx9 = getelementptr %Note, %Note* %.arr, i32 5
store %__pitch* @A4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.5, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.5, i32 0, i32 1)
%lhs_p10 = bitcast %Note* %.idx9 to i8*
call void @memcpy(i8* %lhs_p10, i8* bitcast (%Note* @.litNote.5 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx11 = getelementptr %Note, %Note* %.arr, i32 6
store i32 1, i32* getelementptr inbounds (%__duration, %__duration* @"1/2", i32 0,
    i32 0)
store i32 2, i32* getelementptr inbounds (%__duration, %__duration* @"1/2", i32 0,
    i32 1)
```

```
store %__pitch* @G4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.6, i32 0, i32 0)
store %__duration* @"1/2", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.6, i32 0, i32 1)
%lhs__p12 = bitcast %Note* %.idx11 to i8*
call void @memcpy(i8* %lhs__p12, i8* bitcast (%Note* @.litNote.6 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
store i64 7, i64* getelementptr inbounds (%Arr__Struct__Note, %Arr__Struct__Note*
    @.litarr__Struct__Note, i32 0, i32 0)
store %Note* %.arr, %Note** getelementptr inbounds (%Arr__Struct__Note,
    %Arr__Struct__Note* @.litarr__Struct__Note, i32 0, i32 1)
call void @memcpy(i8* bitcast (%Arr__Struct__Note* @__bt.seq1 to i8*), i8* bitcast
    (%Arr__Struct__Note* @.litarr__Struct__Note to i8*), i64 ptrtoint
    (%Arr__Struct__Note* getelementptr (%Arr__Struct__Note, %Arr__Struct__Note*
    null, i32 1) to i64))
%1 = trunc i64 7 to i32
%mallocsize13 = mul i32 %1, trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr
    (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall14 = tail call i8* @malloc(i32 %mallocsize13)
%.arr15 = bitcast i8* %malloccall14 to %Note*
%.idx16 = getelementptr %Note, %Note* %.arr15, i32 0
store i8 70, i8* getelementptr inbounds (%__pitch, %__pitch* @F4__0, i32 0, i32 0)
store i32 4, i32* getelementptr inbounds (%__pitch, %__pitch* @F4__0, i32 0, i32 1)
store i32 0, i32* getelementptr inbounds (%__pitch, %__pitch* @F4__0, i32 0, i32 2)
store %__pitch* @F4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.7, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.7, i32 0, i32 1)
%lhs__p17 = bitcast %Note* %.idx16 to i8*
call void @memcpy(i8* %lhs__p17, i8* bitcast (%Note* @.litNote.7 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx18 = getelementptr %Note, %Note* %.arr15, i32 1
store %__pitch* @F4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.8, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.8, i32 0, i32 1)
%lhs__p19 = bitcast %Note* %.idx18 to i8*
call void @memcpy(i8* %lhs__p19, i8* bitcast (%Note* @.litNote.8 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx20 = getelementptr %Note, %Note* %.arr15, i32 2
store i8 69, i8* getelementptr inbounds (%__pitch, %__pitch* @E4__0, i32 0, i32 0)
store i32 4, i32* getelementptr inbounds (%__pitch, %__pitch* @E4__0, i32 0, i32 1)
store i32 0, i32* getelementptr inbounds (%__pitch, %__pitch* @E4__0, i32 0, i32 2)
store %__pitch* @E4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.9, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.9, i32 0, i32 1)
%lhs__p21 = bitcast %Note* %.idx20 to i8*
call void @memcpy(i8* %lhs__p21, i8* bitcast (%Note* @.litNote.9 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx22 = getelementptr %Note, %Note* %.arr15, i32 3
store %__pitch* @E4__0, %__pitch** getelementptr inbounds (%Note, %Note*
```

```
    @.litNote.10, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.10, i32 0, i32 1)
%lhs_p23 = bitcast %Note* %.idx22 to i8*
call void @memcpy(i8* %lhs_p23, i8* bitcast (%Note* @.litNote.10 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx24 = getelementptr %Note, %Note* %.arr15, i32 4
store i8 68, i8* getelementptr inbounds (%__pitch, %__pitch* @D4__0, i32 0, i32 0)
store i32 4, i32* getelementptr inbounds (%__pitch, %__pitch* @D4__0, i32 0, i32 1)
store i32 0, i32* getelementptr inbounds (%__pitch, %__pitch* @D4__0, i32 0, i32 2)
store %__pitch* @D4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.11, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.11, i32 0, i32 1)
%lhs_p25 = bitcast %Note* %.idx24 to i8*
call void @memcpy(i8* %lhs_p25, i8* bitcast (%Note* @.litNote.11 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx26 = getelementptr %Note, %Note* %.arr15, i32 5
store %__pitch* @D4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.12, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.12, i32 0, i32 1)
%lhs_p27 = bitcast %Note* %.idx26 to i8*
call void @memcpy(i8* %lhs_p27, i8* bitcast (%Note* @.litNote.12 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx28 = getelementptr %Note, %Note* %.arr15, i32 6
store %__pitch* @C4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.13, i32 0, i32 0)
store %__duration* @"1/2", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.13, i32 0, i32 1)
%lhs_p29 = bitcast %Note* %.idx28 to i8*
call void @memcpy(i8* %lhs_p29, i8* bitcast (%Note* @.litNote.13 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
store i64 7, i64* getelementptr inbounds (%Arr_Struct_Note, %Arr_Struct_Note*
    @.litarr_Struct_Note.14, i32 0, i32 0)
store %Note* %.arr15, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.litarr_Struct_Note.14, i32 0, i32 1)
call void @memcpy(i8* bitcast (%Arr_Struct_Note* @__bt.seq2 to i8*), i8* bitcast
    (%Arr_Struct_Note* @.litarr_Struct_Note.14 to i8*), i64 ptrtoint
    (%Arr_Struct_Note* getelementptr (%Arr_Struct_Note, %Arr_Struct_Note*
    null, i32 1) to i64))
%2 = trunc i64 7 to i32
%mallocsize30 = mul i32 %2, trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr
    (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall31 = tail call i8* @malloc(i32 %mallocsize30)
%.arr32 = bitcast i8* %malloccall31 to %Note*
%.idx33 = getelementptr %Note, %Note* %.arr32, i32 0
store %__pitch* @G4__0, %__pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.15, i32 0, i32 0)
store %__duration* @"1/4", %__duration** getelementptr inbounds (%Note, %Note*
    @.litNote.15, i32 0, i32 1)
%lhs_p34 = bitcast %Note* %.idx33 to i8*
```

```
call void @memcpy(i8* %lhs_p34, i8* bitcast (%Note* @.litNote.15 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx35 = getelementptr %Note, %Note* %.arr32, i32 1
store %_pitch* @G4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.16, i32 0, i32 0)
store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.16, i32 0, i32 1)
%lhs_p36 = bitcast %Note* %.idx35 to i8*
call void @memcpy(i8* %lhs_p36, i8* bitcast (%Note* @.litNote.16 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx37 = getelementptr %Note, %Note* %.arr32, i32 2
store %_pitch* @F4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.17, i32 0, i32 0)
store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.17, i32 0, i32 1)
%lhs_p38 = bitcast %Note* %.idx37 to i8*
call void @memcpy(i8* %lhs_p38, i8* bitcast (%Note* @.litNote.17 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx39 = getelementptr %Note, %Note* %.arr32, i32 3
store %_pitch* @F4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.18, i32 0, i32 0)
store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.18, i32 0, i32 1)
%lhs_p40 = bitcast %Note* %.idx39 to i8*
call void @memcpy(i8* %lhs_p40, i8* bitcast (%Note* @.litNote.18 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx41 = getelementptr %Note, %Note* %.arr32, i32 4
store %_pitch* @E4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.19, i32 0, i32 0)
store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.19, i32 0, i32 1)
%lhs_p42 = bitcast %Note* %.idx41 to i8*
call void @memcpy(i8* %lhs_p42, i8* bitcast (%Note* @.litNote.19 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx43 = getelementptr %Note, %Note* %.arr32, i32 5
store %_pitch* @E4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.20, i32 0, i32 0)
store %_duration* @"1/4", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.20, i32 0, i32 1)
%lhs_p44 = bitcast %Note* %.idx43 to i8*
call void @memcpy(i8* %lhs_p44, i8* bitcast (%Note* @.litNote.20 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
%.idx45 = getelementptr %Note, %Note* %.arr32, i32 6
store %_pitch* @D4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.21, i32 0, i32 0)
store %_duration* @"1/2", %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.21, i32 0, i32 1)
%lhs_p46 = bitcast %Note* %.idx45 to i8*
call void @memcpy(i8* %lhs_p46, i8* bitcast (%Note* @.litNote.21 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
store i64 7, i64* getelementptr inbounds (%Arr_Struct_Note, %Arr_Struct_Note*
    @.litarr_Struct_Note.22, i32 0, i32 0)
```

```
store %Note* %.arr32, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.litarr_Struct_Note.22, i32 0, i32 1)
call void @memcpy(i8* bitcast (%Arr_Struct_Note* @_bt.seq3 to i8*), i8* bitcast
    (%Arr_Struct_Note* @.litarr_Struct_Note.22 to i8*), i64 ptrtoint
    (%Arr_Struct_Note* getelementptr (%Arr_Struct_Note, %Arr_Struct_Note*
    null, i32 1) to i64))
%.arrlen = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq1, i32 0, i32 0))
%.arrlen47 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq2, i32 0, i32 0))
%.arrlen48 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq3, i32 0, i32 0))
%.arrlen49 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq3, i32 0, i32 0))
%.arrlen50 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq1, i32 0, i32 0))
%.arrp = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq2, i32 0, i32 1)
%.rawidx = getelementptr %Note, %Note* %.arrp, i32 0
%.arrlen51 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq2, i32 0, i32 0))
%tmp = sub i32 %.arrlen51, 1
%tmp52 = sub i32 %tmp, 0
%mallocsize53 = mul i32 %tmp52, trunc (i64 mul nuw (i64 ptrtoint (i1**
    getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall54 = tail call i8* @malloc(i32 %mallocsize53)
%.arrsub_p = bitcast i8* %malloccall54 to %Note*
%.lencast = sext i32 %tmp52 to i64
%.size = mul i64 %.lencast, mul nuw (i64 ptrtoint (i1** getelementptr (i1*, i1**
    null, i32 1) to i64), i64 2)
%lhs_p55 = bitcast %Note* %.arrsub_p to i8*
%rhs_p = bitcast %Note* %.rawidx to i8*
call void @memcpy(i8* %lhs_p55, i8* %rhs_p, i64 %.size)
%.lencast56 = sext i32 %tmp52 to i64
store i64 %.lencast56, i64* getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.arrsub, i32 0, i32 0)
store %Note* %.arrsub_p, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.arrsub, i32 0, i32 1)
%.arrlen57 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @.arrsub, i32 0, i32 0))
%.add = add i32 0, %.arrlen
%.add58 = add i32 %.add, %.arrlen47
%.add59 = add i32 %.add58, %.arrlen48
%.add60 = add i32 %.add59, %.arrlen49
%.add61 = add i32 %.add60, %.arrlen50
%.add62 = add i32 %.add61, %.arrlen57
%.add63 = add i32 %.add62, 1
%mallocsize64 = mul i32 %.add63, trunc (i64 mul nuw (i64 ptrtoint (i1**
    getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall65 = tail call i8* @malloc(i32 %mallocsize64)
%.arrconcat_p = bitcast i8* %malloccall65 to %Note*
%.arrp66 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
```

```
    %Arr_Struct_Note* @_bt.seq2, i32 0, i32 1)
%.rawidx67 = getelementptr %Note, %Note* %.arrp66, i32 0
%.arrlen68 = call i32 bitcast (i32 (i8*)* @len to i32 (i64*)*)(i64* getelementptr
    inbounds (%Arr_Struct_Note, %Arr_Struct_Note* @_bt.seq2, i32 0, i32 0))
%tmp69 = sub i32 %.arrlen68, 1
%tmp70 = sub i32 %tmp69, 0
%mallocsize71 = mul i32 %tmp70, trunc (i64 mul nuw (i64 ptrtoint (i1**
    getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall72 = tail call i8* @malloc(i32 %mallocsize71)
%.arrsub_p73 = bitcast i8* %malloccall72 to %Note*
%.lencast74 = sext i32 %tmp70 to i64
%.size75 = mul i64 %.lencast74, mul nuw (i64 ptrtoint (i1** getelementptr (i1*, i1**
    null, i32 1) to i64), i64 2)
%lhs_p76 = bitcast %Note* %.arrsub_p73 to i8*
%rhs_p77 = bitcast %Note* %.rawidx67 to i8*
call void @memcpy(i8* %lhs_p76, i8* %rhs_p77, i64 %.size75)
%.lencast78 = sext i32 %tmp70 to i64
store i64 %.lencast78, i64* getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.arrsub.23, i32 0, i32 0)
store %Note* %.arrsub_p73, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.arrsub.23, i32 0, i32 1)
%3 = trunc i64 1 to i32
%mallocsize79 = mul i32 %3, trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr
    (i1*, i1** null, i32 1) to i64), i64 2) to i32)
%malloccall80 = tail call i8* @malloc(i32 %mallocsize79)
%.arr81 = bitcast i8* %malloccall80 to %Note*
%.idx82 = getelementptr %Note, %Note* %.arr81, i32 0
%_bt.w = load %_duration*, %_duration** @_bt.w
store %_pitch* @C4_0, %_pitch** getelementptr inbounds (%Note, %Note*
    @.litNote.24, i32 0, i32 0)
store %_duration* %_bt.w, %_duration** getelementptr inbounds (%Note, %Note*
    @.litNote.24, i32 0, i32 1)
%lhs_p83 = bitcast %Note* %.idx82 to i8*
call void @memcpy(i8* %lhs_p83, i8* bitcast (%Note* @.litNote.24 to i8*), i64 mul
    nuw (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
store i64 1, i64* getelementptr inbounds (%Arr_Struct_Note, %Arr_Struct_Note*
    @.litarr_Struct_Note.25, i32 0, i32 0)
store %Note* %.arr81, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.litarr_Struct_Note.25, i32 0, i32 1)
%.arrp84 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq1, i32 0, i32 1)
%.rawidx85 = getelementptr %Note, %Note* %.arrp84, i32 0
%.concatidx = getelementptr %Note, %Note* %.arrconcat_p, i32 0
%.lencast86 = sext i32 %.arrlen to i64
%.size87 = mul i64 %.lencast86, mul nuw (i64 ptrtoint (i1** getelementptr (i1*, i1**
    null, i32 1) to i64), i64 2)
%lhs_p88 = bitcast %Note* %.concatidx to i8*
%rhs_p89 = bitcast %Note* %.rawidx85 to i8*
call void @memcpy(i8* %lhs_p88, i8* %rhs_p89, i64 %.size87)
%.arrp90 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq2, i32 0, i32 1)
%.rawidx91 = getelementptr %Note, %Note* %.arrp90, i32 0
```

```
%.concatidx92 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add
%.lencast93 = sext i32 %.arrlen47 to i64
%.size94 = mul i64 %.lencast93, mul nuw (i64 ptrtoint (i1** getelementptr (i1*, i1**
    null, i32 1) to i64), i64 2)
%lhs_p95 = bitcast %Note* %.concatidx92 to i8*
%rhs_p96 = bitcast %Note* %.rawidx91 to i8*
call void @memcpy(i8* %lhs_p95, i8* %rhs_p96, i64 %.size94)
%.arrp97 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq3, i32 0, i32 1)
%.rawidx98 = getelementptr %Note, %Note* %.arrp97, i32 0
%.concatidx99 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add58
%.lencast100 = sext i32 %.arrlen48 to i64
%.size101 = mul i64 %.lencast100, mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i64), i64 2)
%lhs_p102 = bitcast %Note* %.concatidx99 to i8*
%rhs_p103 = bitcast %Note* %.rawidx98 to i8*
call void @memcpy(i8* %lhs_p102, i8* %rhs_p103, i64 %.size101)
%.arrp104 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq3, i32 0, i32 1)
%.rawidx105 = getelementptr %Note, %Note* %.arrp104, i32 0
%.concatidx106 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add59
%.lencast107 = sext i32 %.arrlen49 to i64
%.size108 = mul i64 %.lencast107, mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i64), i64 2)
%lhs_p109 = bitcast %Note* %.concatidx106 to i8*
%rhs_p110 = bitcast %Note* %.rawidx105 to i8*
call void @memcpy(i8* %lhs_p109, i8* %rhs_p110, i64 %.size108)
%.arrp111 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @_bt.seq1, i32 0, i32 1)
%.rawidx112 = getelementptr %Note, %Note* %.arrp111, i32 0
%.concatidx113 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add60
%.lencast114 = sext i32 %.arrlen50 to i64
%.size115 = mul i64 %.lencast114, mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i64), i64 2)
%lhs_p116 = bitcast %Note* %.concatidx113 to i8*
%rhs_p117 = bitcast %Note* %.rawidx112 to i8*
call void @memcpy(i8* %lhs_p116, i8* %rhs_p117, i64 %.size115)
%.arrp118 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.arrsub.23, i32 0, i32 1)
%.rawidx119 = getelementptr %Note, %Note* %.arrp118, i32 0
%.concatidx120 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add61
%.lencast121 = sext i32 %.arrlen57 to i64
%.size122 = mul i64 %.lencast121, mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i64), i64 2)
%lhs_p123 = bitcast %Note* %.concatidx120 to i8*
%rhs_p124 = bitcast %Note* %.rawidx119 to i8*
call void @memcpy(i8* %lhs_p123, i8* %rhs_p124, i64 %.size122)
%.arrp125 = load %Note*, %Note** getelementptr inbounds (%Arr_Struct_Note,
    %Arr_Struct_Note* @.litarr_Struct_Note.25, i32 0, i32 1)
%.rawidx126 = getelementptr %Note, %Note* %.arrp125, i32 0
%.concatidx127 = getelementptr %Note, %Note* %.arrconcat_p, i32 %.add62
%lhs_p128 = bitcast %Note* %.concatidx127 to i8*
```

```
  %rhs_p129 = bitcast %Note* %.rawidx126 to i8*
  call void @memcpy(i8* %lhs_p128, i8* %rhs_p129, i64 mul nuw (i64 ptrtoint (i1**
      getelementptr (i1*, i1** null, i32 1) to i64), i64 2))
  %.lencast130 = sext i32 %.add63 to i64
  store i64 %.lencast130, i64* getelementptr inbounds (%Arr_Struct_Note,
      %Arr_Struct_Note* @.arrconcat, i32 0, i32 0)
  store %Note* %.arrconcat_p, %Note** getelementptr inbounds (%Arr_Struct_Note,
      %Arr_Struct_Note* @.arrconcat, i32 0, i32 1)
  call void @memcpy(i8* bitcast (%Arr_Struct_Note* @_bt.seq to i8*), i8* bitcast
      (%Arr_Struct_Note* @.arrconcat to i8*), i64 ptrtoint (%Arr_Struct_Note*
      getelementptr (%Arr_Struct_Note, %Arr_Struct_Note* null, i32 1) to i64))
  call void @render_as_midi(%Arr_Struct_Note* @_bt.seq)
  ret void
}

declare i32 @printf(i8*, ...)

declare void @memcpy(i8*, i8*, i64)

declare noalias i8* @malloc(i32)

; Function Attrs: nounwind ssp uwtable
define i32 @len(i8* %arr_struct_p) #0 {
entry:
  %arr_struct_p.addr = alloca i8*, align 8
  store i8* %arr_struct_p, i8** %arr_struct_p.addr, align 8
  %0 = load i8*, i8** %arr_struct_p.addr, align 8
  %1 = bitcast i8* %0 to i64*
  %2 = load i64, i64* %1, align 8
  %conv = trunc i64 %2 to i32
  ret i32 %conv
}

; Function Attrs: nounwind ssp uwtable
define i8* @_str_of_pitch(%_pitch* %p) #0 {
entry:
  %p.addr = alloca %_pitch*, align 8
  %_buffer = alloca i8*, align 8
  %c = alloca i8, align 1
  store %_pitch* %p, %_pitch** %p.addr, align 8
  %call = call i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 4)
  store i8* %call, i8** %_buffer, align 8
  store i8 0, i8* %c, align 1
  %0 = load %_pitch*, %_pitch** %p.addr, align 8
  %alter = getelementptr inbounds %_pitch, %_pitch* %0, i32 0, i32 2
  %1 = load i32, i32* %alter, align 4
  %cmp = icmp eq i32 %1, 1
  br i1 %cmp, label %if.then, label %if. else

if .then:                                         ; preds = %entry
  store i8 35, i8* %c, align 1
  br label %if.end4
```

```
if.else:                                          ; preds = %entry
  %2 = load %__pitch*, %__pitch** %p.addr, align 8
  %alter1 = getelementptr inbounds %__pitch, %__pitch* %2, i32 0, i32 2
  %3 = load i32, i32* %alter1, align 4
  %cmp2 = icmp eq i32 %3, -1
  br i1 %cmp2, label %if.then3, label %if.end

if.then3:                                         ; preds = %if.else
  store i8 98, i8* %c, align 1
  br label %if.end

if.end:                                           ; preds = %if.then3, %if.else
  br label %if.end4

if.end4:                                          ; preds = %if.end, %if.then
  %4 = load i8*, i8** %__buffer, align 8
  %5 = load i8*, i8** %__buffer, align 8
  %6 = call i64 @llvm.objectsize.i64.p0i8(i8* %5, i1 false)
  %7 = load %__pitch*, %__pitch** %p.addr, align 8
  %key = getelementptr inbounds %__pitch, %__pitch* %7, i32 0, i32 0
  %8 = load i8, i8* %key, align 4
  %conv = sext i8 %8 to i32
  %9 = load %__pitch*, %__pitch** %p.addr, align 8
  %octave = getelementptr inbounds %__pitch, %__pitch* %9, i32 0, i32 1
  %10 = load i32, i32* %octave, align 4
  %11 = load i8, i8* %c, align 1
  %conv5 = sext i8 %11 to i32
  %call6 = call i32 (i8*, i32, i64, i8*, ...) @___sprintf_chk(i8* %4, i32 0, i64 %6,
      i8* getelementptr inbounds ([7 x i8], [7 x i8]* @.str, i32 0, i32 0), i32 %conv,
      i32 %10, i32 %conv5)
  %12 = load i8*, i8** %__buffer, align 8
  ret i8* %12
}

; Function Attrs: nounwind readnone
declare i64 @llvm.objectsize.i64.p0i8(i8*, i1) #1

declare i32 @___sprintf_chk(i8*, i32, i64, i8*, ...) #2

; Function Attrs: nounwind ssp uwtable
define i8* @_str_of_duration(%__duration* %d) #0 {
entry:
  %d.addr = alloca %__duration*, align 8
  %__buffer = alloca i8*, align 8
  store %__duration* %d, %__duration** %d.addr, align 8
  %call = call i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 10)
  store i8* %call, i8** %__buffer, align 8
  %0 = load i8*, i8** %__buffer, align 8
  %1 = load i8*, i8** %__buffer, align 8
  %2 = call i64 @llvm.objectsize.i64.p0i8(i8* %1, i1 false)
  %3 = load %__duration*, %__duration** %d.addr, align 8
```

```
  %a = getelementptr inbounds %__duration, %__duration* %3, i32 0, i32 0
  %4 = load i32, i32* %a, align 4
  %5 = load %__duration*, %__duration** %d.addr, align 8
  %b = getelementptr inbounds %__duration, %__duration* %5, i32 0, i32 1
  %6 = load i32, i32* %b, align 4
  %call1 = call i32 (i8*, i32, i64, i8*, ...) @___sprintf_chk(i8* %0, i32 0, i64 %2,
      i8* getelementptr inbounds ([6 x i8], [6 x i8]* @.str.1, i32 0, i32 0), i32 %4,
      i32 %6)
  %7 = load i8*, i8** %__buffer, align 8
  ret i8* %7
}

; Function Attrs: nounwind ssp uwtable
define i8* @__str__of__Note(%Note* %note) #0 {
entry:
  %note.addr = alloca %Note*, align 8
  %__buffer = alloca i8*, align 8
  %p = alloca %__pitch*, align 8
  %d = alloca %__duration*, align 8
  store %Note* %note, %Note** %note.addr, align 8
  %call = call i8* bitcast (i8* (i32)* @malloc to i8* (i64)*)(i64 14)
  store i8* %call, i8** %__buffer, align 8
  %0 = load %Note*, %Note** %note.addr, align 8
  %p1 = getelementptr inbounds %Note, %Note* %0, i32 0, i32 0
  %1 = load %__pitch*, %__pitch** %p1, align 8
  store %__pitch* %1, %__pitch** %p, align 8
  %2 = load %Note*, %Note** %note.addr, align 8
  %d2 = getelementptr inbounds %Note, %Note* %2, i32 0, i32 1
  %3 = load %__duration*, %__duration** %d2, align 8
  store %__duration* %3, %__duration** %d, align 8
  %4 = load %__pitch*, %__pitch** %p, align 8
  %alter = getelementptr inbounds %__pitch, %__pitch* %4, i32 0, i32 2
  %5 = load i32, i32* %alter, align 4
  %cmp = icmp eq i32 %5, 1
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %6 = load i8*, i8** %__buffer, align 8
  %7 = load i8*, i8** %__buffer, align 8
  %8 = call i64 @llvm.objectsize.i64.p0i8(i8* %7, i1 false)
  %9 = load %__pitch*, %__pitch** %p, align 8
  %key = getelementptr inbounds %__pitch, %__pitch* %9, i32 0, i32 0
  %10 = load i8, i8* %key, align 4
  %conv = sext i8 %10 to i32
  %11 = load %__pitch*, %__pitch** %p, align 8
  %octave = getelementptr inbounds %__pitch, %__pitch* %11, i32 0, i32 1
  %12 = load i32, i32* %octave, align 4
  %13 = load %__duration*, %__duration** %d, align 8
  %a = getelementptr inbounds %__duration, %__duration* %13, i32 0, i32 0
  %14 = load i32, i32* %a, align 4
  %15 = load %__duration*, %__duration** %d, align 8
  %b = getelementptr inbounds %__duration, %__duration* %15, i32 0, i32 1
```

```
  %16 = load i32, i32* %b, align 4
  %call3 = call i32 (i8*, i32, i64, i8*, ...) @___sprintf_chk(i8* %6, i32 0, i64 %8,
      i8* getelementptr inbounds ([12 x i8], [12 x i8]* @.str.2, i32 0, i32 0), i32
      %conv, i32 %12, i32 %14, i32 %16)
  br label %if.end21

if.else:                                          ; preds = %entry
  %17 = load %__pitch*, %__pitch** %p, align 8
  %alter4 = getelementptr inbounds %__pitch, %__pitch* %17, i32 0, i32 2
  %18 = load i32, i32* %alter4, align 4
  %cmp5 = icmp eq i32 %18, -1
  br i1 %cmp5, label %if.then7, label %if.else14

if.then7:                                         ; preds = %if.else
  %19 = load i8*, i8** %__buffer, align 8
  %20 = load i8*, i8** %__buffer, align 8
  %21 = call i64 @llvm.objectsize.i64.p0i8(i8* %20, i1 false)
  %22 = load %__pitch*, %__pitch** %p, align 8
  %key8 = getelementptr inbounds %__pitch, %__pitch* %22, i32 0, i32 0
  %23 = load i8, i8* %key8, align 4
  %conv9 = sext i8 %23 to i32
  %24 = load %__pitch*, %__pitch** %p, align 8
  %octave10 = getelementptr inbounds %__pitch, %__pitch* %24, i32 0, i32 1
  %25 = load i32, i32* %octave10, align 4
  %26 = load %__duration*, %__duration** %d, align 8
  %a11 = getelementptr inbounds %__duration, %__duration* %26, i32 0, i32 0
  %27 = load i32, i32* %a11, align 4
  %28 = load %__duration*, %__duration** %d, align 8
  %b12 = getelementptr inbounds %__duration, %__duration* %28, i32 0, i32 1
  %29 = load i32, i32* %b12, align 4
  %call13 = call i32 (i8*, i32, i64, i8*, ...) @___sprintf_chk(i8* %19, i32 0, i64
      %21, i8* getelementptr inbounds ([12 x i8], [12 x i8]* @.str.2, i32 0, i32 0), i32
      %conv9, i32 %25, i32 %27, i32 %29)
  br label %if.end

if.else14:                                        ; preds = %if.else
  %30 = load i8*, i8** %__buffer, align 8
  %31 = load i8*, i8** %__buffer, align 8
  %32 = call i64 @llvm.objectsize.i64.p0i8(i8* %31, i1 false)
  %33 = load %__pitch*, %__pitch** %p, align 8
  %key15 = getelementptr inbounds %__pitch, %__pitch* %33, i32 0, i32 0
  %34 = load i8, i8* %key15, align 4
  %conv16 = sext i8 %34 to i32
  %35 = load %__pitch*, %__pitch** %p, align 8
  %octave17 = getelementptr inbounds %__pitch, %__pitch* %35, i32 0, i32 1
  %36 = load i32, i32* %octave17, align 4
  %37 = load %__duration*, %__duration** %d, align 8
  %a18 = getelementptr inbounds %__duration, %__duration* %37, i32 0, i32 0
  %38 = load i32, i32* %a18, align 4
  %39 = load %__duration*, %__duration** %d, align 8
  %b19 = getelementptr inbounds %__duration, %__duration* %39, i32 0, i32 1
  %40 = load i32, i32* %b19, align 4
```

```
  %call20 = call i32 (i8*, i32, i64, i8*, ...) @___sprintf_chk(i8* %30, i32 0, i64
      %32, i8* getelementptr inbounds ([11 x i8], [11 x i8]* @.str.3, i32 0, i32 0), i32
      %conv16, i32 %36, i32 %38, i32 %40)
  br label %if.end

if.end:                                           ; preds = %if.else14, %if.then7
  br label %if.end21

if.end21:                                         ; preds = %if.end, %if.then
  %41 = load i8*, i8** %_buffer, align 8
  ret i8* %41
}

; Function Attrs: nounwind ssp uwtable
define void @__write_sequence_midi_text(i64 %input_sequence.coerce0, %Note*
    %input_sequence.coerce1, i32 %seqi) #0 {
entry:
  %input_sequence = alloca %Arr_Struct_Note, align 8
  %seqi.addr = alloca i32, align 4
  %saved_stack = alloca i8*, align 8
  %file_pointer = alloca %struct.___sFILE*, align 8
  %sentenc = alloca [1000 x i8], align 16
  %cwd = alloca [1000 x i8], align 16
  %i = alloca i32, align 4
  %0 = bitcast %Arr_Struct_Note* %input_sequence to { i64, %Note* }*
  %1 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %0, i32 0, i32 0
  store i64 %input_sequence.coerce0, i64* %1, align 8
  %2 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %0, i32 0, i32 1
  store %Note* %input_sequence.coerce1, %Note** %2, align 8
  store i32 %seqi, i32* %seqi.addr, align 4
  %len = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 0
  %3 = load i64, i64* %len, align 8
  %4 = call i8* @llvm.stacksave()
  store i8* %4, i8** %saved_stack, align 8
  %vla = alloca i32, i64 %3, align 16
  %len1 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 0
  %5 = load i64, i64* %len1, align 8
  %vla2 = alloca float, i64 %5, align 16
  %arraydecay = getelementptr inbounds [1000 x i8], [1000 x i8]* %cwd, i32 0, i32 0
  %call = call i8* @getcwd(i8* %arraydecay, i64 1000)
  %cmp = icmp ne i8* %call, null
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %6 = load %struct.___sFILE*, %struct.___sFILE** @___stdoutp, align 8
  %arraydecay3 = getelementptr inbounds [1000 x i8], [1000 x i8]* %cwd, i32 0, i32 0
  %call4 = call i32 (%struct.___sFILE*, i8*, ...) @fprintf(%struct.___sFILE* %6, i8*
      getelementptr inbounds ([25 x i8], [25 x i8]* @.str.4, i32 0, i32 0), i8*
      %arraydecay3)
  %arraydecay5 = getelementptr inbounds [1000 x i8], [1000 x i8]* %cwd, i32 0, i32 0
```

```
  %call6 = call i8* @___strcat_chk(i8* %arraydecay5, i8* getelementptr inbounds ([38
      x i8], [38 x i8]* @.str.5, i32 0, i32 0), i64 1000) #3
  br label %if.end

if.else:                                          ; preds = %entry
  call void @perror(i8* getelementptr inbounds ([15 x i8], [15 x i8]* @.str.6, i32 0,
      i32 0))
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %7 = load i32, i32* %seqi.addr, align 4
  %cmp7 = icmp eq i32 %7, 0
  br i1 %cmp7, label %if.then8, label %if.else11

if.then8:                                         ; preds = %if.end
  %arraydecay9 = getelementptr inbounds [1000 x i8], [1000 x i8]* %cwd, i32 0, i32 0
  %call10 = call %struct.___sFILE* @"\01_fopen"(i8* %arraydecay9, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @.str.7, i32 0, i32 0))
  store %struct.___sFILE* %call10, %struct.___sFILE** %file_pointer, align 8
  br label %if.end14

if.else11:                                        ; preds = %if.end
  %arraydecay12 = getelementptr inbounds [1000 x i8], [1000 x i8]* %cwd, i32 0, i32 0
  %call13 = call %struct.___sFILE* @"\01_fopen"(i8* %arraydecay12, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @.str.8, i32 0, i32 0))
  store %struct.___sFILE* %call13, %struct.___sFILE** %file_pointer, align 8
  br label %if.end14

if.end14:                                         ; preds = %if.else11, %if.then8
  %8 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %cmp15 = icmp eq %struct.___sFILE* %8, null
  br i1 %cmp15, label %if.then16, label %if.end18

if.then16:                                        ; preds = %if.end14
  %call17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x
      i8]* @.str.9, i32 0, i32 0))
  call void @exit(i32 1) #6
  unreachable

if.end18:                                         ; preds = %if.end14
  store i32 0, i32* %i, align 4
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %if.end18
  %9 = load i32, i32* %i, align 4
  %conv = sext i32 %9 to i64
  %len19 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 0
  %10 = load i64, i64* %len19, align 8
  %cmp20 = icmp ult i64 %conv, %10
  br i1 %cmp20, label %for.body, label %for.end
```

```
for.body:                                         ; preds = %for.cond
  %11 = load i32, i32* %i, align 4
  %idxprom = sext i32 %11 to i64
  %arr = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 1
  %12 = load %Note*, %Note** %arr, align 8
  %arrayidx = getelementptr inbounds %Note, %Note* %12, i64 %idxprom
  %p = getelementptr inbounds %Note, %Note* %arrayidx, i32 0, i32 0
  %13 = load %_pitch*, %_pitch** %p, align 8
  %call22 = call i32 @_get_midi_pitch(%_pitch* %13)
  %14 = load i32, i32* %i, align 4
  %idxprom23 = sext i32 %14 to i64
  %arrayidx24 = getelementptr inbounds i32, i32* %vla, i64 %idxprom23
  store i32 %call22, i32* %arrayidx24, align 4
  %15 = load i32, i32* %i, align 4
  %idxprom25 = sext i32 %15 to i64
  %arr26 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 1
  %16 = load %Note*, %Note** %arr26, align 8
  %arrayidx27 = getelementptr inbounds %Note, %Note* %16, i64 %idxprom25
  %d = getelementptr inbounds %Note, %Note* %arrayidx27, i32 0, i32 1
  %17 = load %_duration*, %_duration** %d, align 8
  %a = getelementptr inbounds %_duration, %_duration* %17, i32 0, i32 0
  %18 = load i32, i32* %a, align 4
  %conv28 = sitofp i32 %18 to double
  %mul = fmul double 4.000000e+00, %conv28
  %19 = load i32, i32* %i, align 4
  %idxprom29 = sext i32 %19 to i64
  %arr30 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 1
  %20 = load %Note*, %Note** %arr30, align 8
  %arrayidx31 = getelementptr inbounds %Note, %Note* %20, i64 %idxprom29
  %d32 = getelementptr inbounds %Note, %Note* %arrayidx31, i32 0, i32 1
  %21 = load %_duration*, %_duration** %d32, align 8
  %b = getelementptr inbounds %_duration, %_duration* %21, i32 0, i32 1
  %22 = load i32, i32* %b, align 4
  %conv33 = sitofp i32 %22 to double
  %div = fdiv double %mul, %conv33
  %conv34 = fptrunc double %div to float
  %23 = load i32, i32* %i, align 4
  %idxprom35 = sext i32 %23 to i64
  %arrayidx36 = getelementptr inbounds float, float* %vla2, i64 %idxprom35
  store float %conv34, float* %arrayidx36, align 4
  br label %for.inc

for.inc:                                          ; preds = %for.body
  %24 = load i32, i32* %i, align 4
  %inc = add nsw i32 %24, 1
  store i32 %inc, i32* %i, align 4
  br label %for.cond

for.end:                                          ; preds = %for.cond
```

```
  store i32 0, i32* %i, align 4
  br label %for.cond37

for.cond37:                                       ; preds = %for.inc46, %for.end
  %25 = load i32, i32* %i, align 4
  %conv38 = sext i32 %25 to i64
  %len39 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 0
  %26 = load i64, i64* %len39, align 8
  %cmp40 = icmp ult i64 %conv38, %26
  br i1 %cmp40, label %for.body42, label %for.end48

for.body42:                                       ; preds = %for.cond37
  %27 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %28 = load i32, i32* %i, align 4
  %idxprom43 = sext i32 %28 to i64
  %arrayidx44 = getelementptr inbounds i32, i32* %vla, i64 %idxprom43
  %29 = load i32, i32* %arrayidx44, align 4
  %call45 = call i32 (%struct.___sFILE*, i8*, ...) @fprintf(%struct.___sFILE* %27, i8*
      getelementptr inbounds ([4 x i8], [4 x i8]* @.str.10, i32 0, i32 0), i32 %29)
  br label %for.inc46

for.inc46:                                        ; preds = %for.body42
  %30 = load i32, i32* %i, align 4
  %inc47 = add nsw i32 %30, 1
  store i32 %inc47, i32* %i, align 4
  br label %for.cond37

for.end48:                                        ; preds = %for.cond37
  %31 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %call49 = call i32 (%struct.___sFILE*, i8*, ...) @fprintf(%struct.___sFILE* %31, i8*
      getelementptr inbounds ([4 x i8], [4 x i8]* @.str.11, i32 0, i32 0))
  store i32 0, i32* %i, align 4
  br label %for.cond50

for.cond50:                                       ; preds = %for.inc60, %for.end48
  %32 = load i32, i32* %i, align 4
  %conv51 = sext i32 %32 to i64
  %len52 = getelementptr inbounds %Arr_Struct_Note, %Arr_Struct_Note*
      %input_sequence, i32 0, i32 0
  %33 = load i64, i64* %len52, align 8
  %cmp53 = icmp ult i64 %conv51, %33
  br i1 %cmp53, label %for.body55, label %for.end62

for.body55:                                       ; preds = %for.cond50
  %34 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %35 = load i32, i32* %i, align 4
  %idxprom56 = sext i32 %35 to i64
  %arrayidx57 = getelementptr inbounds float, float* %vla2, i64 %idxprom56
  %36 = load float, float * %arrayidx57, align 4
  %conv58 = fpext float %36 to double
  %call59 = call i32 (%struct.___sFILE*, i8*, ...) @fprintf(%struct.___sFILE* %34, i8*
```

```
      getelementptr inbounds ([4 x i8], [4 x i8]* @.str.12, i32 0, i32 0), double
      %conv58)
  br label %for.inc60

for.inc60:                                        ; preds = %for.body55
  %37 = load i32, i32* %i, align 4
  %inc61 = add nsw i32 %37, 1
  store i32 %inc61, i32* %i, align 4
  br label %for.cond50

for.end62:                                        ; preds = %for.cond50
  %38 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %call63 = call i32 (%struct.___sFILE*, i8*, ...) @fprintf(%struct.___sFILE* %38, i8*
      getelementptr inbounds ([4 x i8], [4 x i8]* @.str.11, i32 0, i32 0))
  %39 = load %struct.___sFILE*, %struct.___sFILE** %file_pointer, align 8
  %call64 = call i32 @fclose(%struct.___sFILE* %39)
  %40 = load i8*, i8** %saved_stack, align 8
  call void @llvm.stackrestore(i8* %40)
  ret void
}

; Function Attrs: nounwind
declare i8* @llvm.stacksave() #3

declare i8* @getcwd(i8*, i64) #2

declare i32 @fprintf(%struct.___sFILE*, i8*, ...) #2

; Function Attrs: nounwind
declare i8* @___strcat_chk(i8*, i8*, i64) #4

declare void @perror(i8*) #2

declare %struct.___sFILE* @"\01_fopen"(i8*, i8*) #2

; Function Attrs: noreturn
declare void @exit(i32) #5

; Function Attrs: nounwind ssp uwtable
define i32 @_get_midi_pitch(%_pitch* %p) #0 {
entry:
  %p.addr = alloca %_pitch*, align 8
  %note_number_index = alloca i32, align 4
  %note_number = alloca i32, align 4
  store %_pitch* %p, %_pitch** %p.addr, align 8
  store i32 0, i32* %note_number_index, align 4
  %0 = load %_pitch*, %_pitch** %p.addr, align 8
  %key = getelementptr inbounds %_pitch, %_pitch* %0, i32 0, i32 0
  %1 = load i8, i8* %key, align 4
  %conv = sext i8 %1 to i32
  %cmp = icmp eq i32 %conv, 65
  br i1 %cmp, label %if.then, label %if.else
```

```
if.then:                                          ; preds = %entry
  store i32 5, i32* %note_number_index, align 4
  br label %if.end10

if.else:                                          ; preds = %entry
  %2 = load %__pitch*, %__pitch** %p.addr, align 8
  %key2 = getelementptr inbounds %__pitch, %__pitch* %2, i32 0, i32 0
  %3 = load i8, i8* %key2, align 4
  %conv3 = sext i8 %3 to i32
  %cmp4 = icmp eq i32 %conv3, 66
  br i1 %cmp4, label %if.then6, label %if.else7

if.then6:                                         ; preds = %if.else
  store i32 6, i32* %note_number_index, align 4
  br label %if.end

if.else7:                                         ; preds = %if.else
  %4 = load %__pitch*, %__pitch** %p.addr, align 8
  %key8 = getelementptr inbounds %__pitch, %__pitch* %4, i32 0, i32 0
  %5 = load i8, i8* %key8, align 4
  %conv9 = sext i8 %5 to i32
  %sub = sub nsw i32 %conv9, 67
  store i32 %sub, i32* %note_number_index, align 4
  br label %if.end

if.end:                                           ; preds = %if.else7, %if.then6
  br label %if.end10

if.end10:                                         ; preds = %if.end, %if.then
  %6 = load %__pitch*, %__pitch** %p.addr, align 8
  %octave = getelementptr inbounds %__pitch, %__pitch* %6, i32 0, i32 1
  %7 = load i32, i32* %octave, align 4
  %add = add nsw i32 %7, 1
  %mul = mul nsw i32 %add, 12
  %8 = load i32, i32* %note_number_index, align 4
  %idxprom = sext i32 %8 to i64
  %arrayidx = getelementptr inbounds [7 x i32], [7 x i32]* @__pitch_values, i64 0, i64
      %idxprom
  %9 = load i32, i32* %arrayidx, align 4
  %add11 = add nsw i32 %mul, %9
  %10 = load %__pitch*, %__pitch** %p.addr, align 8
  %alter = getelementptr inbounds %__pitch, %__pitch* %10, i32 0, i32 2
  %11 = load i32, i32* %alter, align 4
  %add12 = add nsw i32 %add11, %11
  store i32 %add12, i32* %note_number, align 4
  %12 = load i32, i32* %note_number, align 4
  ret i32 %12
}

declare i32 @fclose(%struct.___sFILE*) #2
```

```
; Function Attrs: nounwind
declare void @llvm.stackrestore(i8*) #3

; Function Attrs: nounwind ssp uwtable
define void @__make_midi_from_midi_text() #0 {
entry:
  %script = alloca i8*, align 8
  store i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str.13, i32 0, i32 0),
      i8** %script, align 8
  %0 = load i8*, i8** %script, align 8
  %call = call i32 @"\01_system"(i8* %0)
  ret void
}

declare i32 @"\01_system"(i8*) #2

; Function Attrs: nounwind ssp uwtable
define void @render_seqs_as_midi(i32 %num, ...) #0 {
entry:
  %num.addr = alloca i32, align 4
  %valist = alloca [1 x %struct.___va_list_tag], align 16
  %i = alloca i32, align 4
  store i32 %num, i32* %num.addr, align 4
  %arraydecay = getelementptr inbounds [1 x %struct.___va_list_tag], [1 x
      %struct.___va_list_tag]* %valist, i32 0, i32 0
  %arraydecay1 = bitcast %struct.___va_list_tag* %arraydecay to i8*
  call void @llvm.va_start(i8* %arraydecay1)
  store i32 0, i32* %i, align 4
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %0 = load i32, i32* %i, align 4
  %1 = load i32, i32* %num.addr, align 4
  %cmp = icmp slt i32 %0, %1
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %arraydecay2 = getelementptr inbounds [1 x %struct.___va_list_tag], [1 x
      %struct.___va_list_tag]* %valist, i32 0, i32 0
  %gp_offset_p = getelementptr inbounds %struct.___va_list_tag,
      %struct.___va_list_tag* %arraydecay2, i32 0, i32 0
  %gp_offset = load i32, i32* %gp_offset_p, align 16
  %fits_in_gp = icmp ule i32 %gp_offset, 40
  br i1 %fits_in_gp, label %vaarg.in_reg, label %vaarg.in_mem

vaarg.in_reg:                                     ; preds = %for.body
  %2 = getelementptr inbounds %struct.___va_list_tag, %struct.___va_list_tag*
      %arraydecay2, i32 0, i32 3
  %reg_save_area = load i8*, i8** %2, align 16
  %3 = getelementptr i8, i8* %reg_save_area, i32 %gp_offset
  %4 = bitcast i8* %3 to %Arr_Struct_Note**
  %5 = add i32 %gp_offset, 8
```

```
  store i32 %5, i32* %gp_offset_p, align 16
  br label %vaarg.end

vaarg.in_mem:                                     ; preds = %for.body
  %overflow_arg_area_p = getelementptr inbounds %struct.___va_list_tag,
      %struct.___va_list_tag* %arraydecay2, i32 0, i32 2
  %overflow_arg_area = load i8*, i8** %overflow_arg_area_p, align 8
  %6 = bitcast i8* %overflow_arg_area to %Arr_Struct_Note**
  %overflow_arg_area.next = getelementptr i8, i8* %overflow_arg_area, i32 8
  store i8* %overflow_arg_area.next, i8** %overflow_arg_area_p, align 8
  br label %vaarg.end

vaarg.end:                                        ; preds = %vaarg.in_mem,
    %vaarg.in_reg
  %vaarg.addr = phi %Arr_Struct_Note** [ %4, %vaarg.in_reg ], [ %6, %vaarg.in_mem
      ]
  %7 = load %Arr_Struct_Note*, %Arr_Struct_Note** %vaarg.addr, align 8
  %8 = load i32, i32* %i, align 4
  %9 = bitcast %Arr_Struct_Note* %7 to { i64, %Note* }*
  %10 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %9, i32 0, i32 0
  %11 = load i64, i64* %10, align 8
  %12 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %9, i32 0, i32 1
  %13 = load %Note*, %Note** %12, align 8
  call void @__write_sequence_midi_text(i64 %11, %Note* %13, i32 %8)
  br label %for.inc

for.inc:                                          ; preds = %vaarg.end
  %14 = load i32, i32* %i, align 4
  %inc = add nsw i32 %14, 1
  store i32 %inc, i32* %i, align 4
  br label %for.cond

for.end:                                          ; preds = %for.cond
  %arraydecay3 = getelementptr inbounds [1 x %struct.___va_list_tag], [1 x
      %struct.___va_list_tag]* %valist, i32 0, i32 0
  %arraydecay34 = bitcast %struct.___va_list_tag* %arraydecay3 to i8*
  call void @llvm.va_end(i8* %arraydecay34)
  call void @__make_midi_from_midi_text()
  ret void
}

; Function Attrs: nounwind
declare void @llvm.va_start(i8*) #3

; Function Attrs: nounwind
declare void @llvm.va_end(i8*) #3

; Function Attrs: nounwind ssp uwtable
define void @render_as_midi(%Arr_Struct_Note* %input_sequence) #0 {
entry:
  %input_sequence.addr = alloca %Arr_Struct_Note*, align 8
  store %Arr_Struct_Note* %input_sequence, %Arr_Struct_Note**
```

```
      %input_sequence.addr, align 8
  %0 = load %Arr_Struct_Note*, %Arr_Struct_Note** %input_sequence.addr, align 8
  %1 = bitcast %Arr_Struct_Note* %0 to { i64, %Note* }*
  %2 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %1, i32 0, i32 0
  %3 = load i64, i64* %2, align 8
  %4 = getelementptr inbounds { i64, %Note* }, { i64, %Note* }* %1, i32 0, i32 1
  %5 = load %Note*, %Note** %4, align 8
  call void @__write_sequence_midi_text(i64 %3, %Note* %5, i32 0)
  call void @__make_midi_from_midi_text()
  ret void
}

attributes #0 = { nounwind ssp uwtable "disable-tail-calls"="false"
    "less-precise-fpmad"="false" "no-frame-pointer-elim"="true"
    "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
    "no-nans-fp-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="core2"
    "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+ssse3"
    "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { nounwind readnone }
attributes #2 = { "disable-tail-calls"="false" "less-precise-fpmad"="false"
    "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
    "no-infs-fp-math"="false" "no-nans-fp-math"="false"
    "stack-protector-buffer-size"="8" "target-cpu"="core2"
    "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+ssse3"
    "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #3 = { nounwind }
attributes #4 = { nounwind "disable-tail-calls"="false" "less-precise-fpmad"="false"
    "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
    "no-infs-fp-math"="false" "no-nans-fp-math"="false"
    "stack-protector-buffer-size"="8" "target-cpu"="core2"
    "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+ssse3"
    "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #5 = { noreturn "disable-tail-calls"="false" "less-precise-fpmad"="false"
    "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
    "no-infs-fp-math"="false" "no-nans-fp-math"="false"
    "stack-protector-buffer-size"="8" "target-cpu"="core2"
    "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+ssse3"
    "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #6 = { noreturn }

!llvm.ident = !{!0}
!llvm.module.flags = !{!1}

!0 = !{!"clang version 3.8.1 (tags/RELEASE_381/final)"}
!1 = !{i32 1, !"PIC Level", i32 2}
```

Listing 6.6: MIDI Fiile contents

```
4d54 6864 0000 0006 0001 0003 0078 4d54
726b 0000 0004 00ff 2f00 4d54 726b 0000
015a 0090 3c40 7880 3c40 0090 3c40 7880
3c40 0090 4340 7880 4340 0090 4340 7880
```

```
4340 0090 4540 7880 4540 0090 4540 7880
4540 0090 4340 8170 8043 4000 9041 4078
8041 4000 9041 4078 8041 4000 9040 4078
8040 4000 9040 4078 8040 4000 903e 4078
803e 4000 903e 4078 803e 4000 903c 4081
7080 3c40 0090 4340 7880 4340 0090 4340
7880 4340 0090 4140 7880 4140 0090 4140
7880 4140 0090 4040 7880 4040 0090 4040
7880 4040 0090 3e40 8170 803e 4000 9043
4078 8043 4000 9043 4078 8043 4000 9041
4078 8041 4000 9041 4078 8041 4000 9040
4078 8040 4000 9040 4078 8040 4000 903e
4081 7080 3e40 0090 3c40 7880 3c40 0090
3c40 7880 3c40 0090 4340 7880 4340 0090
4340 7880 4340 0090 4540 7880 4540 0090
4540 7880 4540 0090 4340 8170 8043 4000
9041 4078 8041 4000 9041 4078 8041 4000
9040 4078 8040 4000 9040 4078 8040 4000
903e 4078 803e 4000 903e 4078 803e 4000
903c 4083 6080 3c40 00ff 2f00 4d54 726b
0000 0004 00ff 2f00
```

6.5 Testing Roles

- Eunice was in charge of setting up the infrastructure for the tests and designed test cases based on what features were completed and reported errors to those working on the compiler.
- Ruonan was a big help with reorganizing how we compiled and linked the beathoven binary which the test script used. She also created the PrettyPrint.
- Jake also wrote tests scripts for many of the analyzer features he worked on and exception tests, and also designed some error test cases.

Chapter 7

Lessons Learned

7.1 Team Members

7.1.1 Eunice

I learned a lot about making large hard tasks concretely "do-able" through implementing things bit by bit. I also learned a lot about bash scripting and re-learned what standard out/err is. I realized OCaml/functional programming is hard but if you can train yourself to follow logic (I mean really, focus, almost becoming that nested variable) success may follow. Also...programming languages are so awesome! Who would have thought those rules that we learned in CS Theory would actually amount to patterns that take in, interpret, execute, and do real things?

7.1.2 Jake

I can now say I know a functional programming language. I finally got to understand git. I feel a bit more confident about this whole coding thing. It was pretty exciting to encounter some moments where theory actually matters. Also, learning about music theory was a huge bonus.

7.1.3 Rodrigo

I learned that if your work depends on you understanding concepts completely unfamiliar to you (such as how a midi file is encoded or how to use a midi library with a script in a different language), you should always verify with your teammates, your TA's and/or your professor. When verifying your work with other people the uncertainty of the topic becomes much more straightforward since you can better determine what you need to learn and do as well as what you don't. Given how much work researching and understanding unfamiliar things entails, this lesson helped me to have a much better quality of work.

7.1.4 Ruonan

At the stage of finishing printing integers, which is a simplified version of MicroC, I thought I got hang of the most important part of a compiler. When I attempted to print variables of string type (yes, the "Hello World" milestone), I realized that we need at least two more components, an SAST, which carries variables' type information, and an analyzer that produces this tree from Ast. But only when I started working on the code generation of structs and arrays to LLVM IR did I realize that there are so many stuff behind a compiler.

Right now, when I am coding in whatever programming languages, I see something new which I took as granted before. All the features, such as syntactic sugar, can require a great amount of work. I also know why arrays in Python behave that way. I came up with some array syntax during implementation of our language, and that's exactly the weird behavior of Python array.

7.1.5 Sona

I learned most of all that you can't do everything in a group project, and oftentimes when you don't have an assigned role you have to make it up as you go along. There is always something to do, you just have to find roles which allow you to help your teammates without inconveniencing them. My advice for future teams is this: actively talk with your teammates and make sure you're on the same page, whether you're further ahead or lagging behind, so that they can help you. You're all in this together, and the long-term effects of this software project aren't just your ability to learn OCaml, but also your ability to be able to lead or follow in a group.

7.2 Advice for Future Teams

- Use GitHub as version control. It allowed us to document our progress, while simultaneously seeing new commits and additional features being assigned to different people. We were able to split up work and keep a healthy pace by using their different planning tools.
- Start the LRM immediately, as well as completing your scanner and parser by the time the LRM is completed. In fact, we would recommend starting your AST as well, otherwise you'll end up doing the majority of the work at the tail end of the class (like we did).
- Read ahead in the slides so you can start your project early. Following Professor Edwards' schedule runs the risk of not finishing the compiler on time. It will also help you avoid any potential complications. For example, if we had known how difficult the implementation of structs in codegen would be, we would have started much earlier. Codegen and linking, which were integral to our project, were covered way too late for us to have completed the project on time if we had mirrored the lecture schedule.

- Try your best to learn OCaml in the first few weeks you're coding. If you code in a functional language style, or pick up that habit, it's really helpful in organizing the structure of your code and compiler.
- Meet regularly at least once a week even if it seems like there's nothing to do. As said by the professor, this project is a marathon, not a sprint. Assign each team member items to complete during the week, then use your weekly group meeting to combine all the different parts, and get everyone on the same page.
- From above: make sure everyone is always on the same page. If one person writes over a couple hundred lines of code, and other people don't know what's happening, people can get lost really easily. This benefits no one, because the person who's done the initial implementation often has to complete the entire implementation of that part (example: semantic checking) if other group members don't understand what they're doing. Keeping everyone up to date on your programming logic will allow people to help you out.

Chapter 8

Code Listing

8.1 Compiler Source Code

Listing 8.1: Makefile

```
# Beathoven: compiler Makefile
# - builds and manages all compiler components

# Make sure ocamlbuild can find opam-managed packages: first run
# eval ‘opam config env‘

# The Caml compilers
OCAMLBUILD = corebuild
OCAMLC = ocamlc
OCAMLLEX = ocamllex
OCAMLYACC = ocamlyacc
OCAMLFLAGS = -use-ocamlfind -pkgs llvm,llvm.bitreader,llvm.linker,yojson
FLAGS = $(OCAMLFLAGS) -cflags -w,+a-4

TARGET = beathoven
CLIB = stdlib

.PHONY :$(TARGET).native

$(TARGET).native :
    @clang-3.8 -c -emit-llvm $(CLIB).c # clang-3.8
    $(OCAMLBUILD) $(FLAGS) $(TARGET).native
    @mv $(TARGET).native $(TARGET)


.PHONY: clean
clean:
    $(OCAMLBUILD) -clean
    rm -f $(TARGET) *.cm[iox] parser.ml parser.mli scanner.ml $(CLIB).bc \
*.out *. diff  *.orig  *.output
```

```
all :
	cd ..; make all
```

Listing 8.2: beathoven.ml

```
(*
 * Authors:
 * - Ruonan Xu
 * - Jake Kwon
 * - Eunice Kokor
 *)

type action = Compile | Help | Raw | Sast

let get_help =
  "Beathoven Usage: beathoven <flag> < [src] > [output_file]\n" ^
  "   -c\tCompile beathoven source file to LLVM IR in output_file with stdlib\n" ^
  "   -h\tDisplay this list of options\n"

(* Error reporting helper function *)
let get_pos_and_tok lexbuf =
  let cur = lexbuf.Lexing.lex_curr_p in
  let line_num = cur.Lexing.pos_lnum and
    column_num = cur.Lexing.pos_cnum - cur.Lexing.pos_bol and
    token = Lexing.lexeme lexbuf in
  line_num, column_num, token


let _ =
  let action =
    if Array.length Sys.argv > 1 then
      List.assoc Sys.argv.(1)
        [("-c", Compile) ; ("-h", Help) ; ("-r", Raw); ("-s", Sast)]
    else Compile
  in
  if action = Help then print_endline get_help
  else
    let lexbuf = Lexing.from_channel stdin in
    try
      let ast = Parser.program Scanner.token lexbuf in
      let sast = Analyzer.analyze_ast ast in
      let _ =
        match action with
        | Sast -> print_string (Yojson.Basic.pretty_to_string (Pprint.json_of_program sast))
        | Raw -> ()
        | Compile ->
          let m = Codegen.codegen_program sast in
          (* Llvm_analysis.assert_valid_module m; *) (* Useful built-in check *)
          print_string (Llvm.string_of_llmodule m)
        | Help -> print_string get_help
      in
      if (!Log.has_failed) then raise (Exceptions.ErrorReportedByLog)
    with
    | Parsing.Parse_error ->
      let line_num, column_num, token = get_pos_and_tok lexbuf in
      let errormsg = "parser error line " ^ string_of_int line_num ^ " at column " ^
          string_of_int column_num ^ ": " ^ token in
```

```
52          Log.error (errormsg)
```

Listing 8.3: ast.ml

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *  - Eunice Kokor
6   *)
7
8  let beathoven_lib = "stdlib.bt"
9  let default_mname = "_bt"
10 let default_fname = "_main"
11
12 type primitive =
13      Unit
14    | Bool
15    | Int
16    | Double
17    | String
18    | Char
19    (* primitive music type *)
20    | Pitch
21    | Duration
22
23 (* Change Musictype(Note) as Structtype("Note") so that can access it like struct   *)
24 (* type musictype = Note *)
25
26 type datatype = Primitive of primitive
27                 (* | Musictype of musictype *)
28                 | Structtype of string | Arraytype of datatype
29
30 (* Musictype(Seq) = Arraytype(seq_ele_type).
31  * Seq is internally Arraytype and has all attributes of arrays.
32  * It's better to define it as one of Arraytypes.
33  *)
34 let seq_ele_type = Structtype("Note")
35
36 type binary_operator =
37      Add | Sub | Mult | Div | Mod | Equal | Neq
38    | Less | Leq | Greater | Geq | And | Or
39
40 type unary_operator = Neg | Not
41
42 type bind = datatype * string
43 (* type formal = Formal of bind | Many of datatype *)
44
45 type struct_decl = {
46   sname : string;
47    fields : bind list;
48 }
49
50 type expr =
51      Id of string
52    | StructField of expr * string
53    | LitBool of bool
54    | LitInt of int
55    | LitChar of char
```

```
56    | LitDouble of float
57    | LitStr of string
58    | LitPitch of char * int * int (* step * octave * alter *)
59    | LitDuration of int * int
60    | LitNote of expr * expr (* pitch * duration *)
61    | Null
62    | Binop of expr * binary_operator * expr
63    | Uniop of unary_operator * expr
64    | Assign of expr * expr
65    | FuncCall of string * expr list
66    | Noexpr
67    | LitSeq of expr list
68    | LitArray of expr list
69    | ArrayIdx of expr * expr
70    | ArraySub of expr * expr * expr
71
72  type stmt =
73      Block of stmt list
74    | Expr of expr
75    | VarDecl of datatype * string * expr
76    | Return of expr
77    | If of expr * stmt * stmt
78    | While of expr * stmt
79    | For of expr * expr * expr * stmt
80    | Break
81    | Continue
82    | Struct of struct_decl
83
84
85  type func_decl = {
86    fname : string;
87    formals : bind list;
88    returnType : datatype;
89    body : stmt list;
90  }
91
92  type btmodule = {
93    mname : string;
94    (* TODO: usr_type Enum *)
95    funcs : func_decl list;
96  }
97
98  type include_list = string list
99
100 type program = btmodule list
```

Listing 8.4: scanner.mll

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *  - Sona Roy
6   *)
7
8  {
9    open Parser
10
11   let get_char str =
```

```
12      let str = (Scanf.unescaped str) in
13      str .[1]
14  }
15
16  let lowercase = ['a'-'z']
17  let uppercase = ['A'-'Z']
18  let letter = lowercase | uppercase
19  let digit = ['0'-'9']
20  let newline = ('\n' | '\r' | "\r\n")
21  let whitespace = [' ' '\t']
22  let separator = ';'
23
24  (* Used for char *)
25  let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
26  let ascii = (' '-'!' '#'-'[' ']'-'~')
27
28  (* Used for float parsing *)
29  let hasint = digit+ '.' digit*
30  let hasfrac = digit* '.' digit+
31  let hasexp = 'e' ('+'? | '-') digit+
32
33  let pitch = ['A'-'G'] ( (['0'-'9'] | "10" ) ('#'|'b')?)?
34  let pitch_relative = ['1'-'7'] ('^'|'_')
35
36
37  (* Regex conflicts are resolved by order *)
38  rule token = parse
39    | newline { token lexbuf }
40    | whitespace { token lexbuf }
41    | separator { SEP }
42    | "/*" { comment lexbuf } (* Comments *)
43    | "//" { comment_oneline lexbuf }
44  (* -------------- Scoping -------------- *)
45    | '(' { LPAREN }
46    | ')' { RPAREN }
47    | '[' { LBRACK }
48    | ']' { RBRACK }
49    | '{' { LBRACE }
50    | '}' { RBRACE }
51  (* -------------- Operators -------------- *)
52    | '+' { PLUS }
53    | '-' { MINUS }
54    | '*' { TIMES }
55    | '\\' { DIVIDE } (* otherwise need to infer what's 1/4, binop or duration *)
56    | '%' { MOD }
57    | '=' { ASSIGN }
58    | "==" { EQ }
59    | "!=" { NEQ }
60    | '<' { LT }
61    | "<=" { LTE }
62    | '>' { GT }
63    | ">=" { GTE }
64    | ',' { COMMA }
65    | '!' { NOT }
66    | "&" { PARALLEL }
67    | "and" { AND }
68    | "or" { OR }
69    | "->" { RARROW }
70    | '/' { SLASH }
```

```
71    | "::" { SCORE_RESOLUTION }
72    | ".." { DOTS }
73    | ':' { COLON }
74    | '.' { DOT }
75    | ''' { APOSTROPHE }
76    | '^' { OCTAVE_RAISE }
77    | '_' { OCTAVE_LOWER }
78    | "=>" { MATCHCASE }
79  (* ------------- Keywords ------------- *)
80    | "unit" { UNIT }
81    | "bool" { BOOL }
82    | "int" { INT }
83    | "double" { DOUBLE }
84    | "char" { CHAR }
85    | "string" { STR }
86    | "Struct" { STRUCT }
87    | "Enum" { ENUM }
88    | "if" { IF }
89    | "else"{ ELSE }
90    | "match" { MATCH }
91    | "while" { WHILE }
92    | "for" { FOR }
93    | "in" { IN }
94    | "range" { RANGE }
95    | "break" { BREAK }
96    | "continue" { CONTINUE }
97    | "func" { FUNC }
98    | "return" { RETURN }
99    | "using" { USING }
100   | "module" { MODULE }
101 (*
102   | "null" { NULL }
103 *)
104   | "true" { LIT_BOOL(true) }
105   | "false" { LIT_BOOL(false) }
106 (* ------------- Music Keywords ------------- *)
107   | "pitch" { PITCH }
108   | "duration" { DURATION }
109   | "Note" { NOTE }
110   | "Chord" { CHORD }
111   | "Seq" { SEQ }
112 (* ------------- Literals ------------- *)
113   | digit+ as lit { LIT_INT(int_of_string lit) }
114   | digit+ as lit ".." { LIT_INT_DOTS(int_of_string lit) }
115   | pitch as lit { LIT_PITCH(lit) }
116   | pitch_relative as lit { LIT_PITCH(
117     (Core.Std.Char.to_string (Char.chr (((int_of_char lit.[0] - 48)+1) mod 7 + 65)))
118     ^ (if lit.[1] = '^' then "5" else "3") ) }
119   | ((hasint | hasfrac) hasexp?) | (digit+ hasexp) as lit { LIT_DOUBLE(float_of_string lit) }
120   | ''' ( ascii | digit | escape ) ''' as str { LIT_CHAR(get_char str) }
121   | '"' (('\\' '"'| [^'"'])* as str) '"' { LIT_STR(Scanf.unescaped str) }
122   | (letter) (letter | digit | '_')* '?' as lit { ID(lit) } (* Identifiers should start with
          letters *)
123   | eof { EOF }
124   | _ as c { raise (Exceptions.Lexing_error("Unknown token '" ^ String.make 1 c ^ "'")) }
125
126 and comment = parse
127     "*/" { token lexbuf }
128   | _ { comment lexbuf }
```

```
129
130 and comment_oneline = parse
131     (newline | eof) { token lexbuf }
132   | _ { comment_oneline lexbuf }
```

Listing 8.5: parser.mly

```
1  /*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *  - Sona Roy
6   */
7
8  %{
9    open Ast
10 %}
11
12 /* Token and type specifications */
13 %token <int> LIT_INT LIT_INT_DOTS
14 %token <bool> LIT_BOOL
15 %token <string> LIT_STR
16 %token <float> LIT_DOUBLE
17 %token <string> ID
18 %token <string> LIT_PITCH
19 %token <char> LIT_CHAR
20 %token UNIT BOOL INT CHAR DOUBLE STR
21 %token STRUCT ENUM
22 %token PITCH DURATION NOTE CHORD SEQ
23 %token ASSIGN
24 %token RETURN SEP EOF
25 %token LPAREN RPAREN LBRACK RBRACK LBRACE RBRACE
26 %token PLUS MINUS TIMES DIVIDE MOD
27 %token EQ NEQ LT LTE GT GTE
28 %token COLON DOT COMMA
29 %token NOT AND OR
30 %token RARROW
31 %token SLASH PARALLEL APOSTROPHE DOTS
32 %token OCTAVE_RAISE OCTAVE_LOWER SCORE_RESOLUTION
33 %token FUNC USING MODULE
34 %token MATCH MATCHCASE
35 %token IF ELSE WHILE FOR IN RANGE BREAK CONTINUE
36
37 /* Precedence (low to high) and Associativity */
38 %nonassoc NOELSE
39 %nonassoc ELSE
40 %right ASSIGN
41 %left COLON
42 %left OR
43 %left AND
44 %left EQ NEQ LT GT LTE GTE
45 %left PLUS MINUS
46 %left TIMES DIVIDE MOD
47 %nonassoc SLASH
48 %right NOT
49 %right RBRACK
50 %left LBRACK DOT
51 %nonassoc DOTS
52 /*
```

```
53 %left OCTAVE_RAISE OCTAVE_LOWER
54 */
55
56
57 /* AST program start */
58 %start program
59 %type <Ast.program> program
60
61 %%
62
63 literal_duration:
64   | LIT_INT SLASH LIT_INT { LitDuration($1, $3) }
65
66 literal_pitch:
67   | LIT_PITCH { LitPitch($1.[0],
68     (if (String.length $1 <= 1) then 4 else (int_of_char $1.[1] - int_of_char '0')),
69     (if (String.length $1 <= 2) then 0 else if $1.[2] = '#' then 1 else -1) ) }
70
71 literal_note_complete: /* Maybe I should parse Note literals in scanner with regex */
72   | LIT_INT_DOTS literal_duration { LitNote(LitInt($1), $2) } /* For 5..1/4 */
73
74 literal_note:
75     literal_note_complete { $1 }
76     /* LitSeq only supports complete literal note */
77   | DOTS literal_duration { LitNote(LitPitch('C', 4, 0), $2) }
78
79 literal:
80   /*| NULL { Null }*/
81   | LIT_BOOL { LitBool($1) }
82   | LIT_INT { LitInt($1) }
83   | LIT_DOUBLE { LitDouble($1) }
84   | LIT_STR { LitStr($1) }
85   | LIT_CHAR { LitChar($1) }
86   /* these are still Primitive() */
87   | literal_pitch { $1 }
88   | literal_duration { $1 }
89   | literal_note { $1 }
90
91 primitive:
92     UNIT { Unit }
93   | INT { Int }
94   | DOUBLE { Double }
95   | STR { String }
96   | BOOL { Bool }
97   | CHAR { Char }
98   /* primitive music types */
99   | PITCH { Pitch }
100  | DURATION { Duration }
101
102 datatype_nonarray:
103     primitive { Primitive($1) }
104  /*| NOTE { Musictype(Note) }*/
105  | NOTE { Structtype("Note") }
106  | SEQ { Arraytype(seq_ele_type) }
107  | STRUCT ID { Structtype($2) }
108
109 datatype:
110     datatype_nonarray { $1 }
111  | datatype_nonarray LBRACK RBRACK { Arraytype($1) }
```

```
112
113 /* ------------------- Expressions -------------------- */
114
115 ids:
116     ID { Id($1) }
117   | ids DOT ID { StructField($1, $3) } /* how about struct.struct.f?? */
118   | ids LBRACK expr RBRACK { ArrayIdx($1, $3) } /* ids?? */
119
120 index_range: /* Python-like array access */
121     expr COLON expr { ($1, $3) }
122   | COLON expr { (LitInt(0), $2) }
123   | expr COLON { ($1, Noexpr) }
124   | COLON { (LitInt(0), Noexpr) }
125
126 expr:
127   | ids { $1 }
128   | literal { $1 }
129   | LIT_INT_DOTS ids { LitNote(LitInt($1), $2) } /* For 5..1/4 */
130   | expr DOTS expr { LitNote($1, $3) }
131   | MINUS expr { Uniop (Neg, $2) }
132   | expr PLUS expr { Binop($1, Add, $3) }
133   | expr MINUS expr { Binop($1, Sub, $3) }
134   | expr TIMES expr { Binop($1, Mult, $3) }
135   | expr DIVIDE expr { Binop($1, Div, $3) }
136   | expr MOD expr { Binop($1, Mod, $3) }
137   | expr EQ expr { Binop($1, Equal, $3) }
138   | expr NEQ expr { Binop($1, Neq, $3) }
139   | expr LT expr { Binop($1, Less, $3) }
140   | expr LTE expr { Binop($1, Leq, $3) }
141   | expr GT expr { Binop($1, Greater, $3) }
142   | expr GTE expr { Binop($1, Geq, $3) }
143   | expr AND expr { Binop($1, And, $3) }
144   | expr OR expr { Binop($1, Or, $3) }
145   | NOT expr { Uniop (Not, $2) }
146   | ID LPAREN expr_list RPAREN { FuncCall($1, $3)}
147   | ids ASSIGN expr { Assign($1, $3) }
148   | ids LBRACK index_range RBRACK { ArraySub($1, fst $3, snd $3) }
149   | LBRACK expr_list RBRACK { LitArray($2) }
150   | LT note_list GT { LitSeq($2) } /* using expr_list will have a lot of conflicts */
151   | LPAREN expr RPAREN { $2 }
152
153 /*LPAREN ids RPAREN*/
154
155 /* ------------------- Note List -------------------- */
156
157 note_list:
158   | note_rev_list { List.rev $1 }
159
160 note:
161     ids { $1 }
162   | LIT_INT { LitNote(LitInt($1), LitDuration(1, 4)) }
163   | literal_pitch { LitNote($1, LitDuration(1, 4)) }
164   | literal_note_complete { $1 }
165   | LIT_INT_DOTS ids { LitNote(LitInt($1), $2) }
166   | LIT_INT DOTS literal_duration { LitNote(LitInt($1), $3) } /* For 5 ..1/4 */
167   | literal_pitch DOTS literal_duration { LitNote($1, $3) }
168   | ids DOTS literal_duration { LitNote($1, $3) }
169   | ids DOTS ids { LitNote($1, $3) }
170   | LIT_INT DOTS ids { LitNote(LitInt($1), $3) }
```

```
171   | literal_pitch  DOTS ids { LitNote($1, $3) }
172   /* TODO: LitInt(), ids are not yet supported for Note */
173   | literal_duration { LitNote(LitPitch('C', 4, 0), $1) }
174
175 note_rev_list:
176     /* nothing */ { [] }
177   | note_rev_list note { $2 :: $1 } /* note that id can be whatever datatype */
178
179 /* ---------------------- Expressions List ---------------------- */
180
181 expr_list:
182     /* nothing */ { [] }
183   | expr_rev_list { List.rev $1 }
184
185 expr_rev_list:
186     expr { [$1] }
187   | expr_rev_list COMMA expr { $3 :: $1 }
188
189 expr_opt:
190     /* nothing */ { Noexpr }
191   | expr { $1 }
192
193
194 /* ---------------------- Statements ---------------------- */
195
196 stmt:
197     expr SEP { Expr($1) }
198   | var_decl { $1 }
199   | RETURN expr SEP { Return($2) }
200   | RETURN SEP { Return(Noexpr) }
201   | LBRACE stmt_list RBRACE { Block($2) }
202   | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([Expr(Noexpr)])) }
203   | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
204   | FOR LPAREN expr_opt SEP expr_opt SEP expr_opt RPAREN stmt { For($3, $5, $7, $9) }
205   | FOR ids IN RANGE LPAREN expr COMMA expr RPAREN stmt
206     { For(Assign($2, $6), Binop($2, Less, $8), Assign($2, Binop($2, Add, LitInt(1))), $10) }
207   | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
208   | BREAK SEP { Break }
209   /*| CONTINUE SEP { Continue }*/
210
211 stmt_list:
212   | stmt_rev_list { List.rev $1 }
213
214 stmt_rev_list:
215     /* nothing */ { [] }
216   | stmt_rev_list stmt { $2 :: $1 }
217
218 var_decl:
219     datatype ID SEP { VarDecl($1, $2, Noexpr) }
220   | datatype ID ASSIGN expr SEP { VarDecl($1, $2, $4) }
221
222 /* ---------------------- Structs ---------------------- */
223
224  field_list :
225   | field_rev_list { List.rev $1 }
226
227  field_rev_list :
228     datatype ID SEP { [($1, $2)] }
229   | field_rev_list  datatype ID SEP { ($2, $3) :: $1 }
```

```
230
231 struct_decl:
232   STRUCT ID LBRACE field_list RBRACE
233   {
234     { sname = $2; fields = $4; }
235   }
236
237 struct_and_stmt:
238     stmt { $1 }
239   | struct_decl { Struct($1) }
240
241 /* ---------------------- Functions ---------------------- */
242
243 formal_list: /* bind list */
244     /* nothing */ { [] }
245   | formal_rev_list { List.rev $1 }
246
247 formal_rev_list:
248     datatype ID { [($1, $2)] }
249   | formal_rev_list COMMA datatype ID { ($3, $4) :: $1 }
250
251 func_decl:
252   FUNC ID LPAREN formal_list RPAREN RARROW datatype LBRACE stmt_list RBRACE
253   {
254     { fname = $2; formals = $4; returnType = $7; body = $9 }
255   }
256
257 mfuncs: /* struct_and_stmt_rev_list (main_func), func_decl_rev_list */
258     /* nothing */ { [], [] }
259   | mfuncs struct_and_stmt { ($2 :: fst $1), snd $1 }
260   | mfuncs func_decl { fst $1, ($2 :: snd $1) }
261
262 mbody:
263     mfuncs
264   {
265     { fname = default_fname; formals = [];
266       returnType = Primitive(Unit); body = List.rev (fst $1) } :: (List.rev (snd $1))
267   }
268
269 /* ---------------------- Modules ---------------------- */
270
271 include_decl:
272   | USING ID SEP { ($2, true) }
273   | MODULE ID SEP { ($2, false) }
274
275 include_rev_list:
276     include_decl { [$1] }
277   | include_rev_list include_decl { $2::$1 }
278
279 btmodule:
280   mbody
281   {
282     { mname = default_mname; funcs = $1 }
283   }
284
285   include_list :
286     | include_rev_list { (beathoven_lib, true) :: (List.rev $1)}
287
288 program:
```

```
289     | btmodule EOF { [$1] }
290     | include_list  btmodule EOF { [$2] }
291
292   /* TODO: right now include_list must be on the top */
293   /* Parser is  stateless  (no memory) */
```

Listing 8.6: sast.ml

```
1   (*
2    * Authors:
3    *  - Ruonan Xu
4    *  - Jake Kwon
5    *  - Sona Roy
6    *  - Eunice Kokor
7    *)
8
9   module A = Ast
10
11  type expr =
12      Id of string * A.datatype
13    | StructField of expr * string * A.datatype (* Id * Id * datatype *)
14    | LitBool of bool
15    | LitInt of int
16    | LitChar of char
17    | LitDouble of float
18    | LitStr of string
19    | LitPitch of char * int * int
20    | LitDuration of int * int
21    | LitNote of expr * expr
22    | Null
23    | Binop of expr * A.binary_operator * expr * A.datatype
24    | Uniop of A.unary_operator * expr * A.datatype
25    | Assign of expr * expr * A.datatype
26    | FuncCall of string * expr list * A.datatype
27    | Noexpr
28    | LitArray of expr list * A.datatype (* element type *)
29    | ArrayConcat of expr list * A.datatype (* type of array *)
30    | ArrayIdx of expr * expr * A.datatype
31    | ArraySub of expr * expr * expr * A.datatype
32
33  type stmt =
34      Block of stmt list
35    | Expr of expr * A.datatype
36    | If of expr * stmt * stmt
37    | While of expr * stmt
38    | For of expr * expr * expr * stmt
39    | Return of expr * A.datatype
40    | Break
41    | Continue
42    | VarDecl of A.datatype * string * expr
43
44
45  type func_decl = {
46    fname : string;
47    formals : A.bind list;
48    returnType : A.datatype;
49    body : stmt list;
50    (* TODO: separate vars from stmt list in analyzer or parser ?? *)
51  }
```

```
52
53 type btmodule = {
54   mname : string;
55   structs: A.struct_decl list; (* global name *)
56   funcs : func_decl list; (* global name *)
57 }
58
59 type program = {
60   btmodules : btmodule list;
61 }
```

Listing 8.7: analyzer.ml

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *  - Eunice Kokor
6   *  - Sona Roy
7   *)
8
9  open Ast
10 module A = Ast
11 module S = Sast
12
13 open Environment
14 open Pprint
15
16 module SS = Set.Make(
17   struct
18     let compare = Pervasives.compare
19     type t = datatype
20   end )
21
22
23 (* ---------------------- SAST Utilities ---------------------- *)
24
25 let get_var_type env s =
26   try StringMap.find s env.var_map
27   with | Not_found ->
28     (if env.ismain then raise (Exceptions.VariableNotDefined s));
29     try
30       let (d, _) = StringMap.find s env.formal_map in d
31     with | Not_found ->
32     (* Note that local variables can overwrite module fields (global variables) *)
33     try StringMap.find (get_global_name env.name s) !(env.btmodule).field_map
34     with | Not_found -> raise (Exceptions.VariableNotDefined s)
35
36 let get_type_from_expr (expr : S.expr) =
37   match expr with
38     Id (_,d) -> d
39   | StructField(_,_,d) -> d
40   | LitBool(_) -> A.Primitive(Bool)
41   | LitInt(_) -> A.Primitive(Int)
42   | LitDouble(_) -> A.Primitive(Double)
43   | LitChar(_) -> A.Primitive(Char)
44   | LitStr(_) -> A.Primitive(String)
45   | LitPitch(_,_,_) -> A.Primitive(Pitch)
46   | LitDuration(_,_) -> A.Primitive(Duration)
```

```
47    | LitNote(_,_) -> A.Structtype("Note")
48    | Null -> A.Primitive(Unit) (* Null -> Primitive(Null_t) *)
49    | Binop(_,_,_,d) -> d
50    | Uniop(_,_,d) -> d
51    | Assign(_,_,d) -> d
52    | FuncCall(_,_,d)-> d
53    | Noexpr -> A.Primitive(Unit)
54    | LitArray(_,d) -> Arraytype(d)
55    | ArrayConcat(_,d) -> d
56    | ArrayIdx(_,_,d) -> d
57    | ArraySub(_,_,_,d) -> d
58
59  let get_stmt_from_expr e =
60    let t = get_type_from_expr e in
61    S.Expr(e, t)
62
63  let check_condition (e : S.expr) =
64    match (get_type_from_expr e) with
65    | A.Primitive(Bool) | A.Primitive(Unit) -> ()
66    | _ -> raise (Exceptions.InvalidConditionType)
67
68  let get_litpitch (sast_expr : S.expr) =
69    match get_type_from_expr sast_expr with
70    | A.Primitive(Pitch) -> sast_expr
71    | A.Primitive(Int) -> (
72        match sast_expr with
73        | LitInt(d) ->
74          if d = 0 then S.LitPitch('H',4,0)
75          (* ((d+4) mod 7) + 62) gives right note for each integer input *)
76          else if d >= 1 && d <= 7 then S.LitPitch(Char.chr (( (d+1) mod 7 + 65)),4,0)
77          else raise (Exceptions.InvalidPitchAssignment "make sure your pitch is with in 0-7")
78        | Id(_) -> sast_expr (* TODO: codegen !! *)
79        | _ -> raise (Exceptions.Impossible "get_litpitch")
80      )
81    | _ -> Log.error "[InvalidPitchAssignment]"; sast_expr
82
83  (* let get_map_size map =
84     StringMap.fold (fun k v i -> i + 1) map 0 *)
85
86  (* ---------- build sast from ast ---------- *)
87
88  let rec build_sast_expr env (expr : A.expr) =
89    match expr with
90    | Id(s) -> env,
91                let s = if env.ismain then (get_global_name env.name s) else s in
92                S.Id(s, get_var_type env s)
93    | StructField(e, f) -> analyze_struct env e f
94    | LitBool(b) -> env, S.LitBool(b)
95    | LitInt(i) -> env, S.LitInt(i)
96    | LitDouble(f) -> env, S.LitDouble(f)
97    | LitChar(c) -> env, S.LitChar(c)
98    | LitStr(s) -> env, S.LitStr(s)
99    | LitPitch(k, o, a) -> env, S.LitPitch(k, o, a)
100   | LitDuration(a, b) -> env, S.LitDuration(a, b)
101   | LitNote(p, d) -> analyze_note env p d
102   | Binop(e1, op, e2) -> analyze_binop env e1 op e2
103   | Uniop(op, e) -> analyze_unop env op e
104   | Assign(e1, e2) -> analyze_assign env e1 e2
105   | FuncCall(s, el) -> (* TODO: Chord::func() ?? *)
```

```
106       analyze_funccall env s el  (* env, FuncCall (s,el,_) *)
107     | Noexpr -> env, S.Noexpr
108     | Null -> env, S.Null
109     | LitSeq(el) -> analyze_seq env el (* LitArray(el', seq_ele_type) *)
110     | LitArray(el) -> analyze_array env el
111     | ArrayIdx(a, e) -> analyze_arrayidx env a e
112     | ArraySub(a, e1, e2) -> analyze_arraysub env a e1 e2
113
114 and build_sast_expr_list env (expr_list:A.expr list) =
115     let helper_expr expr = snd (build_sast_expr env expr) in
116     let sast_expr_list = List.map helper_expr expr_list in
117     (* print_int (get_map_size env.var_map); *)
118     env, sast_expr_list
119
120 and analyze_struct env e f =
121     let _, sast_expr = build_sast_expr env e in
122     let field_bind =
123       let struct_type = get_type_from_expr sast_expr in
124       let struct_decl =
125         match struct_type with
126           Structtype n -> (
127             try StringMap.find n !(env.btmodule).struct_map
128             with | Not_found -> raise(Exceptions.Impossible("analyze_struct")))
129         | _ as d -> raise (Exceptions.ShouldAccessStructType (string_of_datatype d))
130       in
131       try List.find (fun field -> (snd field) = f) struct_decl.fields
132       with | Not_found -> raise(Exceptions.StructFieldNotFound(
133           (string_of_datatype struct_type), f))
134     in
135     env, S.StructField(sast_expr, snd field_bind, fst field_bind)
136
137 and analyze_note env p d =
138     let _, pitch = build_sast_expr env p in
139     let pitch = get_litpitch pitch in
140     let _, duration = build_sast_expr env d in
141     env, S.LitNote(pitch, duration)
142
143 (* ----- Array ----- *)
144
145 and analyze_seq env (expr_list:A.expr list) =
146     let _, sast_expr_list = build_sast_expr_list env expr_list in
147     if List.length sast_expr_list = 0 then
148       env, S.LitArray ([], A.seq_ele_type)
149     else
150       let flattened_sast_expr_list =
151         let flatten_seq l (expr : S.expr) =
152           (* Cast datatype and flatten Seq *)
153           match get_type_from_expr expr with
154           | A.Structtype("Note") -> expr :: l
155           | A.Primitive(Pitch) -> S.LitNote(expr, LitDuration(1, 4)) :: l
156           | A.Primitive(Duration) -> S.LitNote(LitPitch('C', 4, 0), expr) :: l
157           | A.Arraytype(seq_ele_type) -> (
158               match expr with
159               | LitArray(el, _) -> (List.rev el) @ l
160               | _ -> expr :: l
161             )
162           (* Future: Chord *)
163           | _ -> Log.error "[TypeNotMatch] Element of Seq should have Note type"; l
164         in
```

```
165         List.rev (List.fold_left flatten_seq [] sast_expr_list)
166       in
167       env, S.LitArray(flattened_sast_expr_list, seq_ele_type)
168
169   (* TODO: update it *)
170   and analyze_array env (expr_list:A.expr list) =
171     let _, sast_expr_list = build_sast_expr_list env expr_list in
172     if List.length sast_expr_list = 0 then
173       env, S.LitArray([], Primitive(Unit))
174     else
175       let get_ele_type expr =
176         let ele_type = get_type_from_expr expr in
177         match ele_type with
178         | Arraytype(d) -> ele_type, d
179         | _ -> ele_type, ele_type
180       in
181       let ele_type = ref (A.Primitive(Unit)) in
182       let sast_expr_list =
183         let helper_array l (expr : S.expr) =
184           let d, d' = get_ele_type expr in
185           Log.debug ((string_of_datatype d) ^ " and " ^ (string_of_datatype d'));
186           if d' = Primitive(Unit) then l (* expr is [] *)
187           else
188             (if !ele_type = Primitive(Unit) then ele_type := d';
189              (* Note that d' is not Unit *)
190              if d = !ele_type then (
191                expr :: (fst l), snd l)
192              else if d' = !ele_type then
193                (
194                  match expr with
195                  | S.LitArray(el, _) -> (el @ (fst l), snd l)
196                  | _ as e ->
197                    let arrays =
198                      if List.length (fst l) = 0 then snd l
199                      else (S.LitArray(fst l, d') :: snd l)
200                    in
201                    ([] , e :: arrays)
202                )
203              else raise (Exceptions.ArrayTypeNotMatch(string_of_datatype d'))
204             )
205         in
206         let l = List.fold_left helper_array ([], []) (List.rev sast_expr_list) in
207         if List.length (fst l) = 0 then snd l
208         else S.LitArray(fst l, !ele_type) :: snd l
209       in
210       let sast_litarray =
211         if List.length sast_expr_list = 1 then List.hd sast_expr_list
212         else S.ArrayConcat(sast_expr_list, Arraytype(!ele_type))
213       in
214       env, sast_litarray
215
216   and analyze_arrayidx env a e =
217     let _, sast_arr = build_sast_expr env a in
218     let ele_type =
219       match get_type_from_expr sast_arr with
220       | Arraytype(d) -> d
221       | _ as d -> raise (Exceptions.ShouldAccessArray(string_of_datatype d))
222     in
223     let _, idx = build_sast_expr env e in
```

```
224   (* TODO J: check idx is int type *)
225   env, S.ArrayIdx(sast_arr, idx, ele_type)
226
227 and analyze_arraysub env a e1 e2 =
228   let _, sast_arr = build_sast_expr env a in
229   let d = get_type_from_expr sast_arr in
230   let get_sast_index e se =
231     let _, idx = build_sast_expr env e in
232     let t = get_type_from_expr idx in
233     if t = Primitive(Int) then idx
234     else if t = Primitive(Unit) then se
235     else (Log.error "[IndexTypeMismatch]"; idx)
236   in
237   match d with
238   | Arraytype(_) -> (
239       let idx1 = get_sast_index e1 (S.LitInt(0)) and
240       idx2 = get_sast_index e2 (S.FuncCall("len", [sast_arr], Primitive(Int))) in
241       env, S.ArraySub(sast_arr, idx1, idx2, d)
242     )
243   | _ -> raise (Exceptions.ShouldAccessArray(string_of_datatype d))
244
245 (* ----- Operators ----- *)
246
247 and analyze_binop env e1 op e2 = (* -> env, Binop (e1,op,e2,t) *)
248   let _, se1 = build_sast_expr env e1 in
249   let _, se2 = build_sast_expr env e2 in
250   let t1 = get_type_from_expr se1 in
251   let t2 = get_type_from_expr se2 in
252   let get_logical_binop_type se1 se2 op = function
253       (A.Primitive(Bool), A.Primitive(Bool)) -> S.Binop(se1, op, se2, A.Primitive(Bool))
254     | _ -> raise (Exceptions.InvalidBinopExpression "Logical operators only operate on Bool
            types")
255   in
256   let get_sast_equality_binop () =
257     if t1 = t2 then
258       match t1 with
259       | Primitive(Bool) | Primitive(Int) | Primitive(String)
260         -> S.Binop(se1, op, se2, A.Primitive(Bool))
261       (* Equality op not supported for double operands. *)
262       | _ -> raise (Exceptions.InvalidBinopExpression "Equality operation is not supported for
              double type")
263     else raise (Exceptions.InvalidBinopExpression "Equality operator can't operate on different
            types")
264   in
265   let get_comparison_binop_type type1 type2 se1 se2 op =
266     let numerics = SS.of_list [A.Primitive(Int); A.Primitive(Double)]
267     in
268     if SS.mem type1 numerics && SS.mem type2 numerics
269     then S.Binop(se1, op, se2, A.Primitive(Bool))
270     else raise (Exceptions.InvalidBinopExpression "Comparison operators operate on numeric
            types only")
271   in
272   let get_arithmetic_binop_type se1 se2 op = function
273       (A.Primitive(Int), A.Primitive(Double))
274     | (A.Primitive(Double), A.Primitive(Int))
275     | (A.Primitive(Double), A.Primitive(Double)) -> S.Binop(se1, op, se2, A.Primitive(Double))
276     (* | (A.Primitive(Int), A.Primitive(Char_t))
277        | (A.Primitive(Char_t), A.Primitive(Int))
278        | (A.Primitive(Char_t), A.Primitive(Char_t)) -> S.Binop(se1, op, se2,
```

```
                A.Primitive(Char_t))
279       *)
280       | (A.Primitive(Int), A.Primitive(Int)) -> S.Binop(se1, op, se2, A.Primitive(Int))
281
282       | _ -> raise (Exceptions.InvalidBinopExpression "Arithmetic operators don't support these
              types")
283     in
284     env, (
285       match op with
286       | And | Or -> get_logical_binop_type se1 se2 op (t1, t2)
287       | Equal | Neq -> get_sast_equality_binop ()
288       | Less | Leq | Greater | Geq -> get_comparison_binop_type t1 t2 se1 se2 op
289       | Add | Mult | Sub | Div | Mod -> get_arithmetic_binop_type se1 se2 op (t1, t2))
290
291 and analyze_unop env op e = (* -> env, Uniop (op,e,_) *)
292     let check_num_unop t = function
293         Neg -> t
294       | _ -> raise(Exceptions.InvalidUnaryOperation)
295     in
296     let check_bool_unop = function
297         Not -> A.Primitive(Bool)
298       | _ -> raise(Exceptions.InvalidUnaryOperation)
299     in
300     let _, se = build_sast_expr env e in
301     let t = get_type_from_expr se in
302     match t with
303       A.Primitive(Int)
304     | A.Primitive(Double) -> env, S.Uniop(op, se, check_num_unop t op)
305     | A.Primitive(Bool) -> env, S.Uniop(op, se, check_bool_unop op)
306     | _ -> raise(Exceptions.InvalidUnaryOperation)
307
308 and analyze_assign env e1 e2 =
309     let _, lhs = build_sast_expr env e1 in
310     let _, rhs = build_sast_expr env e2 in
311     let t1 = get_type_from_expr lhs in
312     let t2 = get_type_from_expr rhs in
313     let rhs =
314       if t1 = t2 then rhs
315       else
316         match t1, t2 with
317         | Arraytype(d), Arraytype(Primitive(Unit)) -> S.LitArray([], d) (* it means e2 is [] *)
318         | Structtype("Note"), _ -> S.LitNote(get_litpitch rhs, S.LitDuration(1, 4))
319         | _ ->
320           raise (Exceptions.AssignTypeMismatch(string_of_datatype t1, string_of_datatype t2))
321     in
322     env, S.Assign(lhs, rhs, t1)
323
324 and analyze_funccall env s el =
325     let _, sast_el = build_sast_expr_list env el in
326     try
327       let func = StringMap.find s builtin_funcs in
328       env, S.FuncCall(func.fname, sast_el, func.returnType)
329     (* TODO: check builtin funcs *)
330     (* such as, len() only accepts arrays *)
331     with | Not_found ->
332     try
333       let fname = get_global_func_name env.name s in
334       let func = StringMap.find fname !(env.btmodule).func_map in (* ast func *)
335       let check_params (actuals : S.expr list) (formals : A.bind list) =
```

```
336         if List.length actuals = List.length formals (* && *)
337         then
338           let paramList = List.map2 (fun i j -> get_type_from_expr i = fst j) actuals formals in
339           if List.mem false paramList
340           then
341             raise (Exceptions.ParamTypeNotMatch "types of paramater differ")
342           else
343             true
344         else raise (Exceptions.ParamNumberNotMatch "numbers of paramater differ")
345       in
346       if check_params sast_el func.formals
347       then
348         env, S.FuncCall(fname, sast_el, func.returnType)
349       else
350         raise (Exceptions.FuncCallCheckFail "funccall check failed")
351     with | Not_found -> raise (Exceptions.FuncNotFound (env.name, s))
352     (*
353     let actuals = handle_params func.sformals sel in
354     let actuals = handle_params f.formals sel in
355     SCall(fname, actuals, func.sreturnType, 0)
356     SCall(sfname, actuals, f.returnType, index)
357     *)
358
359 let get_sast_structtype env s =
360   let n =
361     (* Builtin Struct *)
362     if s = "Note" then s
363     else get_global_name env.name s
364   in
365   if not (StringMap.mem n !(env.btmodule).struct_map)
366   then raise (Exceptions.UndefinedStructType n)
367   else Structtype(n) (* rename Structtype using sast(global) name *)
368
369 let get_sast_arraytype env d =
370   match d with
371   | Structtype(s) -> Arraytype(get_sast_structtype env s)
372   | _ -> Arraytype(d) (* so far there is no arraytype within arraytype !! *)
373
374 let build_sast_vardecl env t1 s e =
375   let s =
376     if env.ismain then get_global_name env.name s else s
377   in
378   if StringMap.mem s env.var_map || StringMap.mem s env.formal_map
379   then raise (Exceptions.DuplicateVariable s)
380   else
381     let t1 =
382       match t1 with
383       | Primitive(Unit) -> raise (Exceptions.UnitTypeError)
384       | Arraytype(d) -> get_sast_arraytype env d
385       | Structtype(s) -> get_sast_structtype env s
386       | _ -> t1
387     in
388     let _, sast_expr = build_sast_expr env e in
389     let t2 = get_type_from_expr sast_expr in
390     let sast_expr =
391       if (t1 = t2) || (sast_expr = S.Noexpr) then sast_expr
392       else
393         match t1, t2 with
394         (* Cast *)
```

```
395           | Arraytype(d), Arraytype(Primitive(Unit)) ->
396             S.LitArray ([], d) (* it means e is [] *)
397           | Primitive(Pitch), Primitive(Int) -> get_litpitch sast_expr
398           | seq_ele_type, _ -> S.LitNote(get_litpitch sast_expr, LitDuration(1, 4))
399           | _ ->
400             raise (Exceptions.VardeclTypeMismatch(string_of_datatype t1, string_of_datatype
                  t2))
401       in
402       env.var_map <- StringMap.add s t1 env.var_map;
403       if env.ismain then (* add variable to module fields *)
404         !(env.btmodule).field_map <- StringMap.add s t1 !(env.btmodule).field_map;
405       env, S.VarDecl(t1, s, sast_expr)
406
407 let rec build_sast_block env = function
408     [] -> env, S.Block([])
409   | _ as l ->
410       let _, sl = build_sast_stmt_list env l in env, S.Block(sl)
411
412 and build_sast_stmt env (stmt : A.stmt) =
413     match stmt with
414     Block sl -> build_sast_block env sl
415   | Expr e -> let _, se = build_sast_expr env e in env, get_stmt_from_expr se
416   | Return e -> build_sast_return e env
417   | If (e, s1, s2) -> build_sast_if e s1 s2 env
418   | For(e1, e2, e3, s) -> build_sast_for e1 e2 e3 s env
419   | While(e, s) -> build_sast_while e s env
420   | Break -> check_break env (* TODO: Need to check if in right context *)
421   | Continue -> check_continue env (* TODO: Need to check if in right context *)
422   | VarDecl(d, s, e) -> build_sast_vardecl env d s e
423   | Struct _ -> env, S.Expr(Noexpr, A.Primitive(Unit)) (* skip structs *)
424
425 and build_sast_stmt_list env (stmt_list:A.stmt list) =
426     let helper_stmt stmt =
427       let sast_stmt = snd (build_sast_stmt env stmt) in
428       sast_stmt (* env will be updated *)
429     in
430     let sast_stmt_list = List.map helper_stmt stmt_list in
431     (* print_int (get_map_size env.var_map); *)
432     env, sast_stmt_list
433
434 and build_sast_return e env =
435     let _, se = build_sast_expr env e in
436     let t = get_type_from_expr se in
437     if t = env.env_returnType
438     then env, S.Return(se, t)
439     else raise (Exceptions.ReturnTypeMismatch(string_of_datatype t, string_of_datatype
          env.env_returnType))
440
441 and build_sast_if e s1 s2 env =
442     let _, se = build_sast_expr env e in
443     let _, ifbody = build_sast_stmt env s1 in
444     let _, elsebody = build_sast_stmt env s2 in
445     if (get_type_from_expr se) = A.Primitive(Bool)
446     then env, S.If(se, ifbody, elsebody)
447     else raise (Exceptions.InvalidConditionType)
448
449 and build_sast_for e1 e2 e3 s env =
450     let old_val = env.env_in_for in
451     env.env_in_for <- true;
```

```
452     let _, se1 = build_sast_expr env e1 in
453     let _, se2 = build_sast_expr env e2 in
454     let _, se3 = build_sast_expr env e3 in
455     let _, body = build_sast_stmt env s in
456     check_condition se2;
457     env.env_in_for <- old_val;
458     env, S.For(se1, se2, se3, body)
459
460 and build_sast_while e s env =
461     let old_val = env.env_in_while in
462     env.env_in_while <- true;
463     let _, se = build_sast_expr env e in
464     let _, body = build_sast_stmt env s in
465     check_condition se;
466     env.env_in_while <- old_val;
467     env, S.While(se, body)
468
469 and check_break env =
470     if env.env_in_for || env.env_in_while then
471       env, S.Break
472     else
473       raise Exceptions.CannotCallBreakOutsideOfLoop
474
475 and check_continue env =
476     if env.env_in_for || env.env_in_while then
477       env, S.Continue
478     else
479       raise Exceptions.CannotCallContinueOutsideOfLoop
480
481 let check_fbody fbody returnType =
482     let len = List.length fbody in
483     if len = 0 then true else
484       let final_stmt = List.hd (List.rev fbody) in
485       match returnType, final_stmt with
486         A.Primitive(Unit), _    -> true
487       |   _, S.Return(_, _)  -> true
488       |   _                  -> false
489
490 let build_sast_func_decl btmodule_map mname btmodule_env ismain (func:A.func_decl) =
491     let env =
492       let formal_map =
493         let helper_formal map formal =
494           StringMap.add (snd formal) formal map
495         in
496         List.fold_left helper_formal StringMap.empty func.formals
497       in
498       {
499         btmodule_map = btmodule_map;
500         (* immutable in this env *)
501         name = mname; (* current module ?? does it change later ?? *)
502         btmodule = btmodule_env; (* current module *)
503         ismain = ismain;
504         formal_map = formal_map;
505         (* mutable in this env *)
506         var_map = StringMap.empty;
507         env_returnType = func.returnType;
508         env_in_for = false;
509         env_in_while = false;
510       }
```

```
511   in
512   let _, fbody = build_sast_stmt_list env func.body in
513   if check_fbody fbody func.returnType
514   then
515     {
516       S.fname = get_global_func_name mname func.fname;
517       S.formals = func.formals;
518       S.returnType = func.returnType;
519       S.body = fbody;
520     }
521   else
522     raise (Exceptions.CheckFbodyFail "check_fbody fail")
523
524 let build_sast_struct_decl mname btmodule_env struct_decl =
525   let sname = get_global_name mname struct_decl.sname in
526   (* Exceptions.DuplicateFunction *)
527   !btmodule_env.struct_map <- (StringMap.add sname struct_decl
          !btmodule_env.struct_map);
528   {
529     sname = sname;
530     fields = struct_decl.fields; (* TODO: rename struct **bind** with global name !!*)
531   }
532
533 let build_sast btmodule_map (btmodule_list:A.btmodule list) =
534   let build_sast_btmodule btmodule =
535     let btmodule_env = ref (StringMap.find btmodule.mname btmodule_map) in
536     let sast_structs =
537       let sast_structs =
538         let helper_struct_decl struct_rev_list = function
539             Struct struct_decl ->
540             let sast_struct =
541               build_sast_struct_decl btmodule.mname btmodule_env struct_decl
542             in sast_struct::struct_rev_list
543           | _ -> struct_rev_list
544         in
545         let (main_func : A.func_decl) = List.hd btmodule.funcs in
546         List.rev (List.fold_left helper_struct_decl [] main_func.body)
547       in
548       (* Note that there is only one copy of builtin_types in default module
549          of Sast, which will be used in codegen *)
550       if btmodule.mname = default_mname then builtin_types_list @ sast_structs
551       else sast_structs
552     in
553     let sast_funcs =
554       let helper_func_decl ismain func =
555         build_sast_func_decl btmodule_map btmodule.mname btmodule_env ismain func
556       in
557       match btmodule.funcs with
558       | [] -> raise (Exceptions.Impossible "Each module has at least one func (main)")
559       | hd :: tl ->
560         (helper_func_decl true hd) :: (List.map (helper_func_decl false) tl)
561     in
562     {
563       S.mname = btmodule.mname;
564       S.structs = sast_structs;
565       S.funcs = sast_funcs;
566     }
567   in
568   List.map build_sast_btmodule btmodule_list
```

```
569
570 let build_btmodule_map (btmodule_list : A.btmodule list) =
571   let build_btmodule_env map btmodule =
572     let helper_func map func =
573       let fname = get_global_func_name btmodule.mname func.fname in
574       if (StringMap.mem fname map)
575       then raise (Exceptions.DuplicateFunction(fname))
576       else if (StringMap.mem (func.fname) builtin_funcs)
577       then raise (Exceptions.CannotUseBuiltinFuncName(func.fname))
578       else StringMap.add fname func map
579     in
580     StringMap.add btmodule.mname
581       {
582         func_map = List.fold_left helper_func StringMap.empty btmodule.funcs;
583         (* Note that there is only one copy of builtin_types in default module
584            of Sast!! *)
585         struct_map = if btmodule.mname = default_mname then builtin_types
586           else StringMap.empty;
587         field_map = StringMap.empty;
588       }
589       map
590   in
591   List.fold_left build_btmodule_env StringMap.empty btmodule_list
592
593 let analyze_ast (btmodule_list) =
594   let btmodule_map = build_btmodule_map btmodule_list in
595   let sast_btmodule_list = build_sast btmodule_map btmodule_list in
596   {
597     S.btmodules = sast_btmodule_list;
598   }
```

Listing 8.8: pprint.ml

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *)
6
7  (*
8  Pretty Print
9  *)
10
11 open Sast
12 (* Ast is opened as module A in Sast. Items in Ast can be accessed with A.* here. *)
13 open Yojson
14 (* Ref: https://realworldocaml.org/v1/en/html/handling-json-data.html *)
15
16 let rec string_of_datatype (t : A.datatype) =
17   match t with
18   | Primitive(Unit) -> "unit"
19   | Primitive(Bool) -> "bool"
20   | Primitive(Int) -> "int"
21   | Primitive(Double) -> "double"
22   | Primitive(String) -> "string"
23   | Primitive(Char) -> "char"
24   | Primitive(Pitch) -> "pitch"
25   | Primitive(Duration) -> "duration"
26   (* | Musictype(Note) -> "Note" *)
```

```
27    | Structtype(s) -> "Struct_" ^ s
28    | Arraytype(d) -> "Array_" ^ (string_of_datatype d)
29  (* TODO J: other datatypes *)
30
31  let string_of_op (op : A.binary_operator) =
32    match op with
33      Add -> "+"
34    | Sub -> "-"
35    | Mult -> "*"
36    | Div -> "/"
37    | Equal -> "=="
38    | Neq -> "!="
39    | Less -> "<"
40    | Leq -> "<="
41    | Greater -> ">"
42    | Geq -> ">="
43    | And -> "&&"
44    | Mod -> "%"
45    | Or -> "||"
46
47  let string_of_uop (uop : A.unary_operator) =
48    match uop with
49      Neg -> "-"
50    | Not -> "!"
51
52  let rec string_of_expr expr =
53    match expr with
54      LitInt(l) -> string_of_int l
55    | LitBool(true) -> "true"
56    | LitBool(false) -> "false"
57    | Id(s,_) -> s
58    | Binop(e1, o, e2,_) ->
59      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
60    | Uniop(o, e, _) -> string_of_uop o ^ string_of_expr e
61    | Assign(v, e,_) -> string_of_expr v ^ " = " ^ string_of_expr e
62    | FuncCall(f, el, _) ->
63      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
64    | Noexpr -> ""
65    | LitArray(el, _) -> "Array: " ^ (String.concat ", " (List.map string_of_expr el))
66    | LitPitch(_,_,_) -> "pitch expr"
67    | LitNote(_,_) -> "note expr"
68
69
70  (* Print SAST tree representation *)
71
72  let tuple_of_datatype d = ("datatype", `String (string_of_datatype d))
73
74  let rec json_of_expr expr =
75    let (expr_json : Yojson.Basic.json) =
76      match expr with
77        Id(s, d) -> `Assoc [("id", `Assoc [("name", `String s); tuple_of_datatype d])]
78      | StructField(e1, e2, d) -> `Assoc [("StructField",
79                                           `Assoc [("struct", (json_of_expr e1));
80                                                   ("field", `String e2);
81                                                   tuple_of_datatype d;
82                                                  ])]
83      | LitBool(b) -> `Assoc [("bool", `Bool b)]
84      | LitInt(i) -> `Assoc [("int", `Int i)]
85      | LitDouble(d) -> `Assoc [("double", `Float d)]
```

```
86      | LitStr(s) -> `Assoc [("string", `String s)]
87      | LitChar(c) -> `Assoc [("char", `String (Core.Std.Char.to_string c))]
88      | LitPitch(k, o, a) ->
89        let p = (Core.Std.Char.to_string k) ^ (string_of_int o) ^ "_" ^ (string_of_int a) in
90        `Assoc [("pitch", `String p)]
91      | LitDuration(a, b) ->
92        `Assoc [("duration", `String ((string_of_int a) ^ "/" ^ (string_of_int b)))]
93      | LitNote(p, d) -> `Assoc [("Note",
94                                  `Assoc [("pitch", (json_of_expr p));
95                                          ("duration", (json_of_expr d));
96                                         ])]
97      | Binop(e1, op, e2, d) -> `Assoc [("binop",
98                                         `Assoc [("lhs", (json_of_expr e1));
99                                                 ("op", `String (string_of_op op));
100                                                ("rhs", (json_of_expr e2));
101                                                tuple_of_datatype d;
102                                               ]) ;]
103     | Uniop(op, e, d) -> `Assoc [("uniop",
104                                   `Assoc [("op", `String (string_of_uop op));
105                                           ("operand", (json_of_expr e));
106                                           tuple_of_datatype d
107                                          ])]
108     | Assign(e1, e2, d)  -> `Assoc [("assign",
109                                      `Assoc [("lhs", (json_of_expr e1));
110                                              ("rhs", (json_of_expr e2));
111                                              tuple_of_datatype d
112                                             ])]
113     | FuncCall(f, el, d)-> `Assoc [("funccall",
114                                     `Assoc [("name", `String f);
115                                             ("params", `List (List.map json_of_expr el));
116                                             tuple_of_datatype d
117                                            ])]
118     | Noexpr -> `String "noexpr"
119     | Null -> `String "null"
120     (* | LitSeq(el) -> `Assoc [("Seq", `List (List.map json_of_expr el))] *)
121     | LitArray(el, d) -> `Assoc [("LitArray",
122                                   `Assoc [("elements", `List (List.map json_of_expr el));
123                                           tuple_of_datatype d
124                                          ])]
125     | ArrayConcat(el, d) -> `Assoc [("ArrayConcat",
126                                      `Assoc [("arrays", `List (List.map json_of_expr el));
127                                              tuple_of_datatype d
128                                             ])]
129     | ArrayIdx(a, idx, d) -> `Assoc [("ArrayIdx",
130                                       `Assoc [("Array", json_of_expr a);
131                                               ("Idx", json_of_expr idx);
132                                               tuple_of_datatype d
133                                              ])]
134     | ArraySub(a, idx1, idx2, d) -> `Assoc [("ArraySub",
135                                              `Assoc [("Array", json_of_expr a);
136                                                      ("Idx1", json_of_expr idx1);
137                                                      ("Idx2", json_of_expr idx2);
138                                                      tuple_of_datatype d
139                                                     ])]
140   in expr_json
141
142 let rec json_of_stmt stmt =
143   let (stmt_json : Yojson.Basic.json) = (* OCaml cannot infer data type as I wish *)
144     match stmt with
```

```
145       Block sl -> `Assoc [("block", `List (List.map json_of_stmt sl))]
146     | Expr(e, d) -> `Assoc [("stmt_expr", `Assoc [("expr", json_of_expr e); tuple_of_datatype
            d])]
147     | Return(e, d) -> `Assoc [("return", `Assoc [("expr", json_of_expr e); tuple_of_datatype
            d])]
148     | If (e, s1, s2) -> `Assoc [("if", `Assoc [("cond", json_of_expr e); ("then", json_of_stmt
            s1)]); ("else", json_of_stmt s2)]
149     | For (e1, e2, e3, s) -> `Assoc [("for",
150                                       `Assoc [("init", json_of_expr e1);
151                                               ("cond", json_of_expr e2);
152                                               ("next", json_of_expr e3);
153                                               ("body", json_of_stmt s)])]
154     | While (e, s) -> `Assoc [("while", `Assoc [("cond", json_of_expr e); ("body",
            json_of_stmt s)])]
155     | Break -> `String "break"
156     | Continue -> `String "continue"
157     | VarDecl(d, s, e) -> `Assoc [("vardecl",
158                                    `Assoc [tuple_of_datatype d; ("name", `String s); ("val",
                                           json_of_expr e)])]
159   in stmt_json
160
161 let json_of_bind_list bind_list =
162   `List (List.map
163            (function (d, s) -> `Assoc [("name", `String s); tuple_of_datatype d;])
164            bind_list)
165
166 let json_of_func (func : func_decl) =
167   `Assoc[("func_decl",
168           `Assoc[("fname", `String func.fname);
169                  ("returnType", `String (string_of_datatype func.returnType) );
170                  ("formals", json_of_bind_list func.formals);
171                  ("body", `List (List.map json_of_stmt func.body));
172                 ])]
173
174 let json_of_funcs funcs =
175   `List(List.map json_of_func funcs)
176
177 let json_of_struct (s : A.struct_decl) =
178   `Assoc[("struct_decl",
179           `Assoc[("sname", `String s.sname);
180                  ("fields", json_of_bind_list s.fields);
181                 ])]
182
183 let json_of_structs structs =
184   `List(List.map json_of_struct structs)
185
186 let json_of_module btmodule =
187   `Assoc [("btmodule",
188            `Assoc[("mname", `String btmodule.mname);
189                   ("structs", json_of_structs btmodule.structs);
190                   ("funcs", json_of_funcs btmodule.funcs);
191                  ])]
192
193 let json_of_module_list btmodules =
194   `List(List.map json_of_module btmodules)
195
196 let json_of_program program =
197   `Assoc [("program",
198            `Assoc [("btmodules", json_of_module_list program.btmodules);])]
```

Listing 8.9: environment.ml

```
(*
 * Authors:
 *  - Ruonan Xu
 *  - Jake Kwon
 *  - Sona Roy
 *  - Eunice Kokor
 *)

(*
Translation Environments
*)
open Sast
module StringMap = Map.Make (String)


let get_global_func_name mname fname =
  if mname = A.default_mname && fname = A.default_fname
  then "main" (* main entry *)
  (* We use '.' to separate types so llvm will recognize the function name
     and it won't conflict *)
  else mname ^ "." ^ fname

let get_global_name mname n =
  (* TODO: maybe need another module name for user main, instead of
     default_mname. Since user ids are not visible to all. Work on this during
     stdlib.bt *)
  (* if mname = A.default_mname then n else *)
  mname ^ "." ^ n


type btmodule_env = {
  (* an immutable field, as all funcs are known in Ast *)
  func_map : A.func_decl StringMap.t; (* key: global name *)
  mutable struct_map : A.struct_decl StringMap.t; (* key: global name *)
  (* what's the use except findding duplicate?? *)
  mutable field_map : A.datatype StringMap.t; (* key: global name *)
}

(* initialize a new environment for every func *)
type env = {
  (* same for all envs *)
  btmodule_map : btmodule_env StringMap.t;
  (* the module this func is in *)
  name : string;
  btmodule : btmodule_env ref;
  ismain : bool; (* whether this func is main of module *)
  (* func locals *)
  formal_map : A.bind StringMap.t;
  mutable var_map : A.datatype StringMap.t;

  mutable env_returnType: A.datatype; (* why mutable ?? *)
  mutable env_in_for : bool;
  mutable env_in_while : bool;
}

(* Initialize builtin_types. *)
let (builtin_types_list : A.struct_decl list) =
  [{
```

```
59      A.sname = "_pitch";
60      A.fields = [(A.Primitive(Char), "key"); (A.Primitive(Int), "octave");
61                   (A.Primitive(Int), "alter");];
62    };
63    {
64      A.sname = "_duration";
65      A.fields = [(A.Primitive(Int), "a");(A.Primitive(Int), "b");];
66    };
67    {
68      (* TODO: if Note is just struct, why not declare it in stdlib.bt *)
69      A.sname = "Note";
70      A.fields = [(A.Primitive(Pitch), "p");(A.Primitive(Duration), "d");];
71    };]
72
73  let (builtin_types : A.struct_decl StringMap.t) =
74    let add_to_map (builtin_type : A.struct_decl) map =
75      StringMap.add builtin_type.sname builtin_type map
76    in
77    List.fold_right add_to_map builtin_types_list StringMap.empty
78
79  (* Initialize builtin_funcs *)
80  (* This is part is only for function checking, such as parameters type checking *)
81  let (builtin_funcs : func_decl StringMap.t) =
82    let get_func_decl name (returnType : A.datatype) formalsType =
83      {
84        fname = name; body = [];
85        returnType = returnType;
86        (* Note that formal types here correspond to codegen/get_bind_type,
87        codegen_builtin_funcs, and function parameters in stdlib.c *)
88        formals = List.map (fun typ -> (typ, "")) formalsType;
89      }
90    in
91    let unit_t = A.Primitive(Unit) and int_t = A.Primitive(Int)
92    and string_t = A.Primitive(String)
93    and pitch_t = A.Primitive(Pitch) and duration_t = A.Primitive(Duration)
94    in
95    let map = StringMap.empty in
96    let map = StringMap.add "print"
97        (get_func_decl "printf" unit_t []) map in
98    let map = StringMap.add "render_seqs_as_midi"
99        (get_func_decl "render_seqs_as_midi" unit_t []) map in
100   let map = StringMap.add "len"
101       (get_func_decl "len" int_t [ ]) map in (* TODO: add the param *)
102   let map = StringMap.add "render_as_midi"
103       (get_func_decl "render_as_midi" unit_t [ A.Arraytype(A.seq_ele_type) ]) map in (*
              TODO: add the param *)
104   let map = StringMap.add "str_of_pitch"
105       (get_func_decl "_str_of_pitch" string_t [ pitch_t ]) map in
106   let map = StringMap.add "str_of_duration"
107       (get_func_decl "_str_of_duration" string_t [ duration_t ]) map in
108   let map = StringMap.add "str_of_Note"
109       (get_func_decl "_str_of_Note" string_t [ A.Structtype("Note") ]) map in
110   map
```

Listing 8.10: codegen.ml

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
```

```
4   *  - Sona Roy
5   *  - Jake Kwon
6   *  - Eunice Kokor
7   *)
8
9   (*
10  Code generation: translate takes a semantically checked AST and produces LLVM IR
11
12  LLVM tutorial: Make sure to read the OCaml version of the tutorial
13    http://llvm.org/docs/tutorial/index.html
14
15  Detailed documentation on the OCaml LLVM library:
16    http://llvm.moe/
17    http://llvm.moe/ocaml/
18  *)
19
20  module L = Llvm (* LLVM VMCore interface library *)
21  open Sast
22
23  module StringMap = Map.Make(String)
24
25  let _debug = true
26
27  let context = L.global_context () (* global data container *)
28  let the_module = L.create_module context "Beathoven Codegen" (* container *)
29  (* let builder = L.builder context *)
30  let double_t = L.double_type context
31  and i64_t = L.i64_type context
32  and i32_t = L.i32_type context
33  and i8_t = L.i8_type context
34  and i1_t = L.i1_type context
35  and void_t = L.void_type context
36  let str_t = L.pointer_type i8_t
37  let ptr_t = str_t
38  let size_t = L.type_of (L.size_of i8_t)
39  let void_p = L.pointer_type size_t
40  let null_ll = L.const_null i32_t
41  and null_str = L.const_null str_t
42
43  let is_main = ref false
44  (* All vardecls in main adopt global name and
45     are defined as global variables in codegen. *)
46  let global_tbl:(string, L.llvalue) Hashtbl.t = Hashtbl.create 100
47  (* Use our own literal lookup instead of L.lookup_global *)
48  let literal_tbl :(string, L.llvalue) Hashtbl.t = Hashtbl.create 100
49
50  let local_tbl:(string, L.llvalue) Hashtbl.t = Hashtbl.create 50
51  (* In formal_tbl are the actual values of parameters. If need to modify
52     primitives in it, should create a copy variable with the same name
53     in local_tbl.
54  *)
55  let formal_tbl:(string, L.llvalue) Hashtbl.t = Hashtbl.create 10
56
57  let array_tbl:(A.datatype, L.lltype) Hashtbl.t = Hashtbl.create 10
58  let struct_tbl:(string, L.lltype) Hashtbl.t = Hashtbl.create 10
59  let struct_field_indexes:(string, int) Hashtbl.t = Hashtbl.create 50
60  let is_struct_packed = false
61
62
```

```
63 (* ---------- Utils ---------- *)
64
65 let lookup_struct sname =
66   try Hashtbl.find struct_tbl sname
67   with | Not_found -> raise(Exceptions.UndefinedStructType sname)
68
69 let lookup_func fname =
70   match (L.lookup_function fname the_module) with
71   | None -> raise (Exceptions.Impossible "Analyzer should catch undefined funcs")
72   | Some f -> f
73
74 let rec lltype_of_datatype (d : A.datatype) =
75   match d with
76   | Primitive(Unit) -> void_t
77   | Primitive(Int) -> i32_t
78   | Primitive(Double) -> double_t
79   | Primitive(String) -> str_t
80   | Primitive(Bool) -> i1_t
81   | Primitive(Char) -> i8_t
82   | Primitive(Duration) -> L.pointer_type (lookup_struct "_duration")
83   | Primitive(Pitch) -> L.pointer_type (lookup_struct "_pitch")
84   (* | Musictype(Note) -> lookup_struct "Note" *)
85   | Structtype(s) -> lookup_struct s
86   | Arraytype(d) -> lookup_array d
87   | _ -> raise(Exceptions.Impossible("lltype_of_datatype"))
88
89 (* Create the struct for Arraytype(d), {int size; d* ptr; } *)
90 and lookup_array (d : A.datatype) =
91   try Hashtbl.find array_tbl d
92   with | Not_found ->
93     let struct_t = L.named_struct_type context ("Arr_" ^ (Pprint.string_of_datatype d)) in
94     let type_array = [|size_t; L.pointer_type (lltype_of_datatype d)|] in
95     L.struct_set_body struct_t type_array is_struct_packed;
96     Hashtbl.add array_tbl d struct_t;
97     struct_t
98
99 let get_bind_type d =
100   let lltype = lltype_of_datatype d in
101   match d with
102   | Primitive(_) -> lltype
103   | _ -> L.pointer_type lltype
104
105 let lltype_of_bind_list (bind_list : A.bind list) =
106   List.map (fun (d, _) -> get_bind_type d) bind_list
107
108
109 (* Declare local variable and remember its llvalue in local_tbl *)
110 let codegen_local_allocate (typ : A.datatype) var_name builder =
111   if _debug then Log.debug ("codegen_local_allocate: " ^ var_name);
112   let t = lltype_of_datatype typ in
113   let alloca = L.build_alloca t var_name builder in
114   Hashtbl.add local_tbl var_name alloca;
115   alloca
116 (* TODO: L.build_array_alloca *)
117
118 (* Declare global variable and remember its llvalue
119    in global_tbl if it's not a temporary variable *)
120 let codegen_global_allocate (typ : A.datatype) var_name builder isPermanent =
121   Log.debug ("codegen_global_allocate: " ^ var_name);
```

```
122   let  zeroinitializer  = L.const_null (lltype_of_datatype typ) in
123   let  alloca = L.define_global var_name zeroinitializer the_module in
124   if  isPermanent then Hashtbl.add global_tbl var_name alloca;
125   alloca
126
127 let  codegen_allocate (typ : A.datatype) var_name builder =
128   if  _debug then Log.debug ("codegen_allocate: " ^ var_name);
129   if  !is_main then codegen_global_allocate typ var_name builder true
130   else  codegen_local_allocate typ var_name builder
131
132
133 (* Return the value for a variable or formal argument *)
134 (* duplicate name in formal will be overwritten by local *)
135 let  load_id id builder =
136   match id with
137   | Id(s, d) -> (
138       let  isloaded = ref  false  in
139       let  v =
140         try  Hashtbl.find  local_tbl  s
141         with | Not_found ->
142         try
143           let  v = Hashtbl.find formal_tbl s in
144           isloaded := true; v
145         with | Not_found ->
146         try  Hashtbl.find  global_tbl s
147         with Not_found -> raise (Exceptions.Impossible
148                                    ("Undefined var not caught in Analyzer unless there is bug
                                           in Codegen"))
149       in
150       match d with (* Only load primitives *)
151       | A.Primitive(_) -> if !isloaded then v else  L.build_load v s builder
152       | _ -> v )
153   | _ -> raise (Exceptions.Impossible("load_id"))
154
155 let  lookup_id id builder =
156   match id with
157   | Id(s, d) -> (
158       try  Hashtbl.find  local_tbl  s
159       with | Not_found ->
160       try
161         let  v = Hashtbl.find formal_tbl s in
162         let  alloca =
163           if  _debug then Log.debug ("lookup_id (formal_tbl): " ^ s);
164           codegen_allocate d s builder
165         in
166         ignore  (L.build_store v alloca  builder);
167         alloca
168       with | Not_found ->
169       try  Hashtbl.find  global_tbl  s
170       with Not_found -> raise (Exceptions.VariableNotDefined s))
171   | _ -> raise (Exceptions.Impossible("lookup_id"))
172
173
174 let  codegen_lit_alloca isPermanent (typ : A.datatype) var_name builder =
175   let  lltype  = (* Actual type of  literals  *)
176     match typ with
177     | Primitive(Duration) -> lookup_struct "_duration"
178     | Primitive(Pitch)  -> lookup_struct "_pitch"
179     | _ -> lltype_of_datatype typ
```

```
180   in
181   let  zeroinitializer  = L.const_null lltype in
182   let  alloca  = L.define_global var_name zeroinitializer the_module in
183   if  isPermanent then Hashtbl.add literal_tbl var_name alloca;
184   (* TODO: temp using L.build_alloca *)
185   alloca  (* ref  lltype *)
186
187 let  get_lit_alloca isPermanent name d (l : (string * L.llvalue)  list )  builder  =
188   let  alloca  = codegen_lit_alloca isPermanent d name builder in
189   let  set_struct_field  i  ( field ,  llvalue ) =
190     let  field ' = L.build_struct_gep alloca i (name ^ "." ^ field )  builder  in
191     ignore(L.build_store  llvalue  field '  builder)
192   in
193   List . iteri  set_struct_field  l;
194   alloca
195
196 (* These  literals  have unique id and will  be stored  in  literal_tbl  *)
197 let  get_literal_alloca  name d (l :  (string  * L. llvalue )  list )  builder  =
198   try Hashtbl.find  literal_tbl  name
199   with | Not_found ->
200     get_lit_alloca  true  name d l builder
201 (* An alternative  is  to use  initializer  (but need a table  for  initializer  , )
202    L.const_named_struct (lookup_struct "pitch")
203    ([| L.const_null str_t; L.const_int i32_t o; L.const_int i32_t a|]) *)
204 (* Such as @p = global %struct._pitch { i8 67, i32 1, i32 1 }, align 4 *)
205
206 let  get_ids_tmp = function
207   | Id(s, _) -> "." ^ s
208   | StructField(_, s, _) -> ".struct." ^ s
209
210
211 (* ---------- LLVM Utils ---------- *)
212
213 let  break_block = ref (null_ll)
214 (* let  continue_block = ref (null_ll) *)
215 let  in_loop = ref  false
216
217 let  add_terminal builder' f =
218   match L.block_terminator (L.insertion_block builder') with
219   | Some ll -> Log.debug ("add_terminal: " ^ (L.string_of_llvalue ll))
220   | None -> ignore (f builder') (* Add a terminal, i.e. a branch *)
221
222 let  memcpy (lhs : L.llvalue)  (rhs  :  L. llvalue )  (size  :  L. llvalue )  builder  =
223   let  lhs_p = L.build_bitcast lhs ptr_t "lhs_p" builder in
224   let  rhs_p = L.build_bitcast rhs ptr_t "rhs_p" builder in
225   Log.debug ("memcpy(): \n" ^ (L.string_of_llvalue lhs_p) ^ "\n" ^
226              (L.string_of_llvalue  rhs_p) ^ "\n" ^ (L.string_of_llvalue  size));
227   ignore(L.build_call  (lookup_func "memcpy") [|lhs_p; rhs_p; size |] ""  builder);
228   lhs_p
229
230 let  codegen_arraycopy lhs rhs d len builder = (* L.llvalue  *)
231   let  len_cast = L.build_intcast len size_t ".lencast"  builder  in
232   let  size_ele  = L.size_of d in
233   let  size  = L.build_mul len_cast size_ele ".size"  builder  in
234   memcpy lhs rhs size builder
235
236 let  get_array_ptr arr_s idx_ll builder =
237   let  arr_p =
238     let  ptr_p = L.build_struct_gep arr_s 1 (".arrp_p") builder in
```

```
239      L.build_load ptr_p ".arrp" builder
240    in
241    L.build_gep arr_p [| idx_ll |] ".rawidx" builder
242
243  (* ---------------------------------------- *)
244
245  let codegen_pitch k o a builder =
246    let pitch = (Core.Std.Char.to_string k) ^ (string_of_int o) ^ "_" ^ (string_of_int a) in
247    let ptr_lit = get_literal_alloca pitch (A.Primitive(Pitch))
248        [("key", L.const_int i8_t (Char.code k)); ("octave", L.const_int i32_t o);
249         ("alter", L.const_int i32_t a)] builder in
250    ptr_lit (* primitive: _pitch* *)
251
252  let codegen_duration a b builder =
253    let gcd' =
254      let rec gcd a b = if b = 0 then a else gcd b (a mod b) in
255      gcd a b
256    in
257    let a = a / gcd' and b = b / gcd' in
258    let duration = (string_of_int a) ^ "/" ^ (string_of_int b) in
259    let ptr_lit = get_literal_alloca duration (A.Primitive(Duration))
260        [("a", L.const_int i32_t a); ("b", L.const_int i32_t b)] builder
261    in
262    ptr_lit (* primitive: _duration* *)
263  (* Seems there is no need to cast, since when assign we simply store it. *)
264  (* cast_literal_alloca duration (A.Primitive(Duration)) ptr_lit builder *)
265
266  (* ----- Functions ----- *)
267
268  let rec codegen_print expr_list builder =
269    let (llval_expr_list : L.llvalue list) = List.map (codegen_expr builder) expr_list in
270    let printfmt =
271      let llval_and_fmt_of_expr expr = (* -> fmt : string *)
272        let print_fmt_of_datatype (t : A.datatype) =
273          match t with
274          | Primitive(Int) -> "%d"
275          | Primitive(String) -> "%s"
276          | Primitive(Char) -> "%c"
277          | Primitive(Bool) ->
278            (* print_endline (L.string_of_llvalue (List.nth llval_expr_list !idx)); *)
279            "%d" (* TODO: print "true" or "false" *)
280          | Primitive(Double) -> "%lf"
281          | _ -> raise (Exceptions.InvalidTypePassedToPrint)
282        in
283        print_fmt_of_datatype (Analyzer.get_type_from_expr expr)
284      in
285      let fmt_list = List.map llval_and_fmt_of_expr expr_list in
286      let fmt_str = String.concat "" fmt_list in
287      L.build_global_stringptr fmt_str "fmt" builder
288    in
289    let actuals = Array.of_list (printfmt :: llval_expr_list) in
290    L.build_call (lookup_func "printf") actuals "tmp" builder
291    (*
292    ref Dice:
293    let zero = const_int i32_t 0 in
294    let s = build_in_bounds_gep llstrfmt [| zero |] "tmp" llbuilder in
295    build_call printf (Array.of_list (s :: params)) "tmp" llbuilder
296     *)
297
```

```
298
299 and codegen_len el builder =
300   if List.length el <> 1 then (Log.error "[ParamNumberNotMatch]"; null_ll)
301   else (
302     let arr_struct_p = codegen_expr builder (List.hd el) in
303     let arr_struct_p = L.build_pointercast arr_struct_p void_p ".void_p" builder in
304     L.build_call (lookup_func "len") [| arr_struct_p |] ".arrlen" builder)
305
306 and codegen_funccall fname el d builder =
307   let f = lookup_func fname in
308   let (actuals : L.llvalue array) = Array.of_list (List.map (codegen_expr builder) el) in
309   (if _debug then
310      Log.debug ("codegen_funccall(" ^ fname ^ "): ");
311    let helper ll = Log.debug ("- " ^ L.string_of_llvalue ll) in
312    Array.iter helper actuals);
313   match d with
314   | A.Primitive(A.Unit) -> L.build_call f actuals "" builder
315   | _ -> L.build_call f actuals "tmp" builder
316
317 (* ----- Assignment ----- *)
318
319 and codegen_assign_with_lhs lhs rhs_expr builder =
320   let store rhs =
321     ignore(L.build_store rhs lhs builder);
322     rhs
323   in
324   let copy rhs = (* rhs is non-primitive, so rhs is ref *)
325     let size_ll = (* the size of the type which rhs_p points to *)
326       let codegen_sizeof e builder =
327         let lltype = lltype_of_datatype (Analyzer.get_type_from_expr e) in
328         let size_ll = L.size_of lltype in
329         Log.debug ("rhs_size: " ^ (L.string_of_llvalue size_ll));
330         size_ll
331       in
332       codegen_sizeof rhs_expr builder
333     in
334     (* set the value of what lhs_p points_to *)
335     memcpy lhs rhs size_ll builder (* rhs_p *)
336   in
337   let d = Analyzer.get_type_from_expr rhs_expr in
338   let rhs = codegen_expr builder rhs_expr in
339   Log.debug ("lhs: " ^ (L.string_of_llvalue lhs) ^ "\n rhs: " ^ (L.string_of_llvalue rhs));
340   match d with
341   | Primitive(_) -> store rhs
342   | _ -> copy rhs
343
344 and codegen_assign lhs_expr rhs_expr builder =
345   let lhs = codegen_expr_ref builder lhs_expr in
346   codegen_assign_with_lhs lhs rhs_expr builder
347
348 (* ----- Struct ----- *)
349
350 and codegen_note pitch duration builder =
351   let p = codegen_expr builder pitch and d = codegen_expr builder duration in
352   get_lit_alloca false ".litNote" (A.Structtype("Note")) [("p", p); ("d", d)] builder
353
354 and codegen_structfield struct_expr fid isref builder =
355   let struct_ll = codegen_expr builder struct_expr in
356   let tmp_name = fid in (* TODO: get_ids_tmp *)
```

```
357   let field_index = (* TODO: separate *)
358     let field =
359       let global_field_name = function
360         | A.Structtype(s) -> s ^ "." ^ tmp_name
361         | _ -> raise (Exceptions.Impossible("Must be structtype unless Analyzer fails"))
362       in
363       global_field_name (Analyzer.get_type_from_expr struct_expr)
364     in
365     Hashtbl.find struct_field_indexes field
366   in
367   let p = L.build_struct_gep struct_ll field_index tmp_name builder in
368   if isref then p
369   else L.build_load p tmp_name builder
370
371 (* ----- Array ----- *)
372
373 and codegen_raw_array el d builder = (* d is element_type *)
374   let len = L.const_int size_t (List.length el) in
375   (* garbage!! no GC *)
376   let arr = L.build_array_malloc (lltype_of_datatype d) len ".arr" builder in
377   List.iteri (fun i e ->
378       let ptr = L.build_gep arr [| (L.const_int i32_t i) |] ".idx" builder in
379       ignore(codegen_assign_with_lhs ptr e builder);
380     ) el;
381   len, arr (* return llvalue of ptr of element_type *)
382
383 and codegen_set_array_struct d name (len : L.llvalue) (arr : L.llvalue) builder =
384   let alloca = codegen_global_allocate d name builder false in
385   Log.debug ("codegen_set_array_struct: " ^ (L.string_of_llvalue alloca));
386   let arr_len = L.build_struct_gep alloca 0 (name ^ ".len") builder in
387   let arr_p = L.build_struct_gep alloca 1 (name ^ ".p") builder in
388   let len_cast = L.build_intcast len size_t ".lencast" builder in
389   ignore(L.build_store len_cast arr_len builder);
390   ignore(L.build_store arr arr_p builder);
391   alloca
392
393 and codegen_array el d builder =
394   if d = A.Primitive(Unit) then null_ll (* TODO: null!! skip unknown empty array [] *)
395   else
396     let len, arr = codegen_raw_array el d builder in
397     let lit_name = ".litarr_" ^ (Pprint.string_of_datatype d) in
398     codegen_set_array_struct (A.Arraytype(d)) lit_name len arr builder
399
400 and codegen_arrayidx a idx d isref builder =
401   let idx_ll = codegen_expr builder idx in
402   let arr_s = codegen_expr builder a in
403   (* TODO: check idx in range *)
404   let p = get_array_ptr arr_s idx_ll builder in
405   if isref then p
406   else
407     match d with
408     | A.Primitive(_) -> L.build_load p ".val" builder
409     | _ -> p
410
411 and codegen_arraysub a idx1 idx2 d builder =
412   (* TODO: [:], [1:], [:-1] *)
413   let ele_type =
414     match d with
415     | A.Arraytype(d) -> d
```

```
416    in
417    let ele_lltype = lltype_of_datatype ele_type in
418    let arr_idx1 = codegen_arrayidx a idx1 ele_type true builder in
419    (* TODO: check idx1 < idx2 *)
420    let new_len = codegen_binop idx2 A.Sub idx1 builder in
421    let new_arr = L.build_array_malloc ele_lltype new_len ".arrsub_p" builder in
422    (* copy original array elements to new array *)
423    let _ = codegen_arraycopy new_arr arr_idx1 ele_lltype new_len builder in
424    codegen_set_array_struct d ".arrsub" new_len new_arr builder
425
426  and codegen_concat_array el d builder =
427    (* garbage!! no GC *)
428    let len_list = (* len of each array *)
429      let get_len expr =
430        let len =
431          match expr with
432          | LitArray(el, _) -> L.const_int i32_t (List.length el)
433          | _ -> codegen_len [expr] builder
434          (* L.build_bitcast () size_t ".cast" builder *)
435        in
436        Log.debug ("codegen_concat_array: " ^ (L.string_of_llvalue len)); len
437      in
438      List.map get_len el
439    in
440    let acc = ref (L.const_int i32_t 0) in
441    let acc_len_array = (* start position of each array in the new array *)
442      let acc_len len =
443        let old_acc = !acc in
444        Log.debug (L.string_of_llvalue len);
445        acc := L.build_add !acc len ".add" builder;
446        old_acc
447      in
448      Array.of_list (List.map acc_len len_list)
449    in
450    let ele_lltype =
451      let ele_type = match d with
452        | A.Arraytype(d) -> d
453        | _ -> raise (Exceptions.Impossible "Analyzer assures that this is Arraytype" )
454      in
455      lltype_of_datatype ele_type
456    in
457    let new_arr = L.build_array_malloc ele_lltype !acc ".arrconcat_p" builder in
458    let expr_array = Array.of_list (List.map (codegen_expr builder) el) in
459    Log.debug ("codegen_concat_array: expr " ^ (L.string_of_llvalue (expr_array.(0))));
460    (List.iteri (fun i len ->
461          let arr_p = get_array_ptr (expr_array.(i)) (L.const_int i32_t 0) builder in
462          let ptr = L.build_gep new_arr [| acc_len_array.(i) |] ".concatidx" builder in
463          Log.debug ("codegen_concat_array: ptr " ^ (L.string_of_llvalue ptr));
464          ignore(codegen_arraycopy ptr arr_p ele_lltype len builder)
465      ) len_list);
466    codegen_set_array_struct d ".arrconcat" !acc new_arr builder
467
468  (* ----- Operators ----- *)
469
470  and codegen_binop e1 (op : A.binary_operator) e2 builder =
471    let e1' = codegen_expr builder e1
472    and e2' = codegen_expr builder e2 in
473    (match op with
474       Add -> L.build_add
```

```
475     | Sub -> L.build_sub
476     | Mult -> L.build_mul
477     | Div -> L.build_sdiv
478     | Equal -> L.build_icmp L.Icmp.Eq
479     (* TODO: string type equality *)
480     | Neq -> L.build_icmp L.Icmp.Ne
481     | Less -> L.build_icmp L.Icmp.Slt
482     | Leq -> L.build_icmp L.Icmp.Sle
483     | Greater -> L.build_icmp L.Icmp.Sgt
484     | Geq -> L.build_icmp L.Icmp.Sge
485     | And -> L.build_and
486     | Mod -> L.build_srem
487     | Or -> L.build_or
488   ) e1' e2' "tmp" builder
489
490 and codegen_unop (op : Sast.A.unary_operator) e1 builder =
491   let e1' = codegen_expr builder e1 in
492   (match op with
493     | Neg -> L.build_neg
494     | Not -> L.build_not) e1' "tmp" builder
495
496 (* ----- Expressions ----- *)
497
498 (* Construct code for an expression; return its llvalue.
499    For non-primitive type, the returned llvalue is ref.
500 *)
501 and codegen_expr builder = function
502   | Id(_, _) as id -> load_id id builder
503   | StructField(e, f, _) -> codegen_structfield e f false builder (* load *)
504   | LitBool b -> L.const_int i1_t (if b then 1 else 0)
505   | LitInt i -> L.const_int i32_t i
506   | LitDouble d -> L.const_float double_t d
507   | LitStr s -> L.build_global_stringptr s "tmp" builder
508   | LitChar c -> L.const_int i8_t (Char.code c)
509   | LitPitch(k, o, a) -> codegen_pitch k o a builder (* load *)
510   | LitDuration(a, b) -> codegen_duration a b builder (* load *)
511   | LitNote(p, d) -> codegen_note p d builder (* ref *)
512   | Noexpr -> null_ll
513   | Null -> null_ll
514   | Assign(e1, e2, _) -> codegen_assign e1 e2 builder
515   | FuncCall(fname, el, d) ->
516     (match fname with
517       | "printf" -> codegen_print el builder
518       | "len" -> codegen_len el builder
519       | _ -> codegen_funccall fname el d builder )
520   | Binop(e1, op, e2, _) -> codegen_binop e1 op e2 builder
521   | Uniop(op, e1, _) -> codegen_unop op e1 builder
522   (* Note that in Analyzer all legal types will be converted to Note types in a LitSeq *)
523   | LitArray(el, d) -> codegen_array el d builder (* ref *)
524   | ArrayIdx(a, idx, d) -> codegen_arrayidx a idx d false builder (* load *)
525   | ArraySub(a, idx1, idx2, d) -> codegen_arraysub a idx1 idx2 d builder (* ref *)
526   | ArrayConcat(el, d) -> codegen_concat_array el d builder (* ref *)
527
528 and codegen_expr_ref builder expr =
529   match expr with
530   (* Structtype, Arraytype, pitch, duration *)
531   | Id(_, _) -> lookup_id expr builder
532   | StructField(e, f, _) -> codegen_structfield e f true builder
533   | ArrayIdx(a, idx, d) -> codegen_arrayidx a idx d true builder
```

```
534   | _ -> raise (Exceptions.ExpressionNotAssignable(Pprint.string_of_expr expr))
535
536
537 (* ----- Statements ----- *)
538
539 let rec codegen_stmt builder = function
540   | Block sl -> List.fold_left codegen_stmt builder sl
541   | Expr(e, _) -> ignore(codegen_expr builder e); builder
542   | Return(e, d) -> ignore(codegen_ret d e builder); builder
543   | VarDecl(d, s, e) ->
544     ignore(codegen_allocate d s builder);
545     if e <> Noexpr then ignore(codegen_assign (Id(s, d)) e builder);
546     builder
547   | Return(e, d) -> ignore(codegen_ret d e builder); builder
548   | If(e, s1, s2) -> codegen_if e s1 s2 builder
549   | While(pred, body) -> codegen_while pred body builder
550   | For(e1, e2, e3, body) -> codegen_stmt builder (* this way not works well with Continue *)
551                              (Block [
552                                  Expr(e1, Analyzer.get_type_from_expr e1);
553                                  While(e2,
554                                        Block [body;
555                                               Expr(e3, Analyzer.get_type_from_expr e1)]) ] )
556   | Break -> ignore(L.build_br (L.block_of_value !break_block) builder); builder
557   (* | Continue -> ignore(L.build_br (L.block_of_value !continue_block) builder); builder *)
558   | _ -> Core.Std.failwith "[Impossible] Struct declaration are skiped in analyzer"
559
560 and codegen_ret d expr builder =
561   match expr with
562     Noexpr -> L.build_ret_void builder
563   | _ -> L.build_ret (codegen_expr builder expr) builder
564
565 and codegen_if condition sast_then (sast_else : stmt) builder =
566   let cond_val = codegen_expr builder condition in
567   let start_bb = L.insertion_block builder in
568   let the_function = L.block_parent start_bb in
569   (* Insert blocks *)
570   let then_bb = L.append_block context "if_then" the_function in
571   let else_bb = L.append_block context "if_else" the_function in
572   let merge_bb = L.append_block context "if_merge" the_function in
573   (* Build if_cond block *)
574   let cond_builder = L.builder_at_end context start_bb in
575   let _ = L.build_cond_br cond_val then_bb else_bb cond_builder in
576   (* Build if_then block *)
577   let then_builder = codegen_stmt (L.builder_at_end context then_bb) sast_then in
578   let _ = add_terminal then_builder (L.build_br merge_bb) in
579   (* Build if_else block *)
580   let else_builder = codegen_stmt (L.builder_at_end context else_bb) sast_else in
581   let _ = add_terminal else_builder (L.build_br merge_bb) in
582   L.builder_at_end context merge_bb
583
584 and codegen_while condition body builder =
585   let the_function = L.block_parent (L.insertion_block builder) in
586   (* Insert blocks *)
587   let cond_bb = L.append_block context "loop_cond" the_function in
588   let body_bb = L.append_block context "loop_body" the_function in
589   let merge_bb = L.append_block context "loop_merge" the_function in
590   (* br label %loop_cond *)
591   let _ = L.build_br cond_bb builder in
592   (* let old_val = !in_loop in
```

```
593     let _ = if not old_val then break_block := L.value_of_block merge_bb in (* break outmost
            loop?? *)
594     let _ = in_loop := true in *)
595     let _ = break_block := L.value_of_block merge_bb in
596     (* Build loop_cond block *)
597     let cond_builder = L.builder_at_end context cond_bb in
598     let cond_val = codegen_expr cond_builder condition in
599     let _ = L.build_cond_br cond_val body_bb merge_bb cond_builder in
600     (* Build loop_body block *)
601     let body_builder = codegen_stmt (L.builder_at_end context body_bb) body in
602     add_terminal body_builder (L.build_br cond_bb);
603     (* in_loop := old_val; *)
604     L.builder_at_end context merge_bb
605
606   let codegen_builtin_funcs () =
607     (* Declare printf(), which the print built-in function will call *)
608     let printf_t = L.var_arg_function_type i32_t [| str_t |] in
609     let _ = L.declare_function "printf" printf_t the_module in
610     let memcpy_t = L.function_type void_t [| ptr_t; ptr_t; size_t |] in
611     let _ = L.declare_function "memcpy" memcpy_t the_module in
612     (* Functions defined in stdlib.bc *)
613     let render_seqs_t = L.var_arg_function_type void_t [| i32_t |] in
614     let _ = L.declare_function "render_seqs_as_midi" render_seqs_t the_module in
615     let len_t = L.function_type i32_t [| void_p |] in
616     let _ = L.declare_function "len" len_t the_module in
617     let render_as_midi_t = L.function_type void_t [| get_bind_type
            (A.Arraytype(A.seq_ele_type)) |] in (* TODO: add param *)
618     let _ = L.declare_function "render_as_midi" render_as_midi_t the_module in
619     let _str_of_pitch_t = L.function_type str_t [| get_bind_type (A.Primitive(Pitch)) |] in
620     let _ = L.declare_function "_str_of_pitch" _str_of_pitch_t the_module in
621     let _str_of_duration_t = L.function_type str_t [| get_bind_type (A.Primitive(Duration)) |]
            in
622     let _ = L.declare_function "_str_of_duration" _str_of_duration_t the_module in
623     let _str_of_Note_t = L.function_type str_t [| get_bind_type (A.Structtype("Note")) |] in
624     let _ = L.declare_function "_str_of_Note" _str_of_Note_t the_module in
625     ()
626
627   let codegen_def_func func =
628     let formals_lltype = lltype_of_bind_list func.formals in
629     let func_t = L.function_type (get_bind_type func.returnType) (Array.of_list formals_lltype)
            in
630     ignore(L.define_function func.fname func_t the_module) (* llfunc *)
631
632   let codegen_func func =
633     Log.debug ("codegen_func: " ^ func.fname);
634     Hashtbl.clear formal_tbl;
635     Hashtbl.clear local_tbl;
636     let init_params llfunc formals =
637       List.iteri ( fun i formal ->
638           let n = snd formal in
639           let v = L.param llfunc i in
640           L.set_value_name n v;
641           Hashtbl.add formal_tbl n v
642         ) formals
643     in
644     let llfunc = lookup_func func.fname in
645     (* An instance of the IRBuilder class used in generating LLVM instructions *)
646     let llbuilder = L.builder_at_end context (L.entry_block llfunc) in
647     let _ = init_params llfunc func.formals in
```

```
648   let llbuilder = codegen_stmt llbuilder (Block(func.body)) in
649   (* Finish off the function. *)
650   if func.returnType = A.Primitive(A.Unit)
651   then ignore(L.build_ret_void llbuilder)
652   else ()
653 (* TODO: return 0 for main. *)
654 (* L.build_ret (L.const_int i32_t 0) llbuilder; *)
655
656 let codegen_def_struct (s : A.struct_decl) =
657   let struct_t = L.named_struct_type context s.sname in
658   Hashtbl.add struct_tbl s.sname struct_t
659
660 let codegen_struct (s : A.struct_decl) =
661   List.iteri (fun i field ->
662       let n = s.sname ^ "." ^ (snd field) in
663       Hashtbl.add struct_field_indexes n i;
664     ) s.fields;
665   let struct_t = lookup_struct s.sname in
666   let type_list = lltype_of_bind_list s.fields in
667   L.struct_set_body struct_t (Array.of_list type_list) is_struct_packed
668 (* TODO: test forward declaration *)
669 (*
670   let llar = [| i32_t; i1_t
671               (* array_type i8_type 10; vector_type i64_type 10 *)
672               |] in
673 *)
674
675 let linker filename =
676   (* let llctx = L.global_context () in *)
677   let llmem = L.MemoryBuffer.of_file filename in
678   let llm = Llvm_bitreader.parse_bitcode context llmem in
679   Llvm_linker.link_modules' the_module llm
680
681 let codegen_program program =
682   let btmodules = program.btmodules in
683   let def_funcs_and_structs btmodule =
684     List.iter codegen_def_struct btmodule.structs;
685     List.iter codegen_def_func btmodule.funcs
686   in
687   let build_funcs_and_structs btmodule =
688     List.iter codegen_struct btmodule.structs;
689     match btmodule.funcs with
690     | [] -> raise (Exceptions.Impossible "Each module has at least one func (main)")
691     | hd :: tl ->
692       (* main of modules *)
693       is_main := true; codegen_func hd;
694       (* functions in modules *)
695       is_main := false; List.iter codegen_func tl
696   in
697   List.iter def_funcs_and_structs btmodules; (* define language structs first *)
698   codegen_builtin_funcs ();
699   List.iter build_funcs_and_structs btmodules; (* main ?? *)
700   linker "stdlib.bc";
701   the_module
```

Listing 8.11: log.ml

```
1 (*
2  * Original version:
```

```
3   * https://github.com/el2724/note-hashtag/blob/master/src/log.ml
4   *)
5
6   type log_level = Error | Warn | Info | Debug
7
8   let min_level = Debug
9   let has_failed = ref false
10
11  let int_of_level = function
12    | Error -> 4
13    | Warn -> 3
14    | Info -> 2
15    | Debug -> 1
16
17  let string_of_level (level : log_level) =
18    match level with
19    | Error -> "error"
20    | Warn -> "warning"
21    | Info -> "info"
22    | Debug -> "debug"
23
24  type color = Bold | Reset | Black | Red | Green | Yellow | Blue | Magenta | Cyan | White
25
26  (* let color_of_level = function *)
27  let color_of_level (level : log_level) =
28    match level with
29    | Error -> Red
30    | Warn -> Yellow
31    | Info -> Blue
32    | Debug -> Cyan
33
34  let string_of_color color =
35    let escape_of_color = function
36      | Reset   -> 0
37      | Bold    -> 1
38      | Black   -> 30
39      | Red     -> 31
40      | Green   -> 32
41      | Yellow  -> 33
42      | Blue    -> 34
43      | Magenta -> 35
44      | Cyan    -> 36
45      | White   -> 37
46    in
47    Printf.sprintf "\027[%dm" (escape_of_color color)
48
49  let print (level : log_level) (fmt : string) =
50    let prefix = (string_of_color (color_of_level level)) ^ (string_of_color Bold) ^
51                 (string_of_level level) ^ ":" ^ (string_of_color Reset) in
52    let printer =
53      if int_of_level level >= int_of_level min_level
54      then Printf.eprintf else (Printf.ifprintf stderr) in
55    printer "%s %s\n%!" prefix fmt
56    (* some info about format 6:
57       [Gagallium : The 6 parameters of '(a, 'b, 'c, 'd, 'e, 'f)
             format6](http://gallium.inria.fr/blog/format6/)
58    *)
59
60  let error fmt = has_failed:= true ; print Error fmt
```

```
61 let warn fmt = print Warn fmt
62 let info fmt = print Info fmt
63 let debug fmt = print Debug fmt
```

Listing 8.12: exceptions.ml

```
1  (*
2   * Authors:
3   *  - Ruonan Xu
4   *  - Jake Kwon
5   *)
6
7  exception Impossible of string
8  exception ErrorReportedByLog
9
10 (* ---------- Scanner ---------- *)
11 exception Lexing_error of string
12
13 (* ---------- Analyzer ---------- *)
14 exception ReturntypeNotMatch of string
15 exception FuncCallCheckFail of string
16 exception CheckFbodyFail of string
17 exception ReturnTypeMismatch of string * string
18 exception CannotCallBreakOutsideOfLoop
19 exception CannotCallContinueOutsideOfLoop
20 exception InvalidUnaryOperation
21
22 (* Variables *)
23 exception VariableNotDefined of string
24 exception DuplicateVariable of string
25 (* Vardecl *)
26 exception UnitTypeError
27 exception VardeclTypeMismatch of string * string
28 exception InvalidPitchAssignment of string
29 (* Operators *)
30 exception InvalidBinopExpression of string
31 (* Assign *)
32 exception AssignTypeMismatch of string * string
33 (* Functions *)
34 exception DuplicateFunction of string
35 exception CannotUseBuiltinFuncName of string
36 exception FuncNotFound of string * string
37 exception ParamNumberNotMatch of string
38 exception ParamTypeNotMatch of string
39 (* Array *)
40 exception ArrayTypeNotMatch of string
41 exception ShouldAccessArray of string
42 (* Struct :: Testing DONE *)
43 exception UndefinedStructType of string
44 exception ShouldAccessStructType of string
45 exception StructFieldNotFound of string * string
46 (* Statement *)
47 exception InvalidConditionType
48
49 (* ---------- Codegen :: Testing DONE ---------- *)
50 (* TODO: Only runtime error should be here. *)
51 exception LLVMFunctionNotFound of string
52 exception InvalidTypePassedToPrint
```

```
53 exception ExpressionNotAssignable of string
```

Listing 8.13: beathoven.h

```
1  /*
2   * Authors:
3   *  - Ruonan Xu
4   */
5
6  /*
7  1. Structs with names starting with '_' are invisible to users.
8  2. struct Part and its fields are visible to users.
9  */
10
11 /* Basic types */
12
13 typedef char * string;
14 typedef void * ptr_t;
15
16 // Arraytype(Int)
17 typedef struct Arr_int {
18     size_t len;
19     int* arr;
20 } Arr_int;
21
22 /* Basic music types */
23
24 // Musictype(Pitch)
25 typedef struct _pitch {
26     char key; // Rest: LitPitch('H',_,_)
27     int octave;
28     int alter;
29 } _pitch;
30 typedef const _pitch * pitch; // since _pitch is literal
31
32
33 // Musictype(Duration)
34 typedef struct _duration {
35     int a;
36     int b;
37 } _duration;
38 typedef const _duration * duration;
39
40
41 // Musictype(Note)
42 typedef struct Note {
43     pitch p;
44     duration d;
45 } Note;
46
47 /*
48 // Skip these types
49 typedef struct Chord {
50     int len;
51     Note* notes;
52     // or
53     //Note notes[4];
54 } Chord;
55
```

```
56 typedef struct _note_or_chord {
57     int type;
58     union {
59         Note *note;
60         Chord *chord;
61     } p;
62     // or
63     // union {
64     //     Note note;
65     //     Chord chord;
66     // } ele;
67     // But, avoid pointers unless it's inevitable
68 } _Seq_ele;
69
70 */
71
72 /* Composite music types */
73
74 // Musictype(Seq)
75 typedef struct Seq {
76     size_t len;
77     // _Seq_ele *arr; // the terrible version
78     Note *arr;
79 } Seq;
80 typedef Seq Arr_Note;
81
82 typedef struct _Sequence {
83     Seq seq;
84     double startTime;
85     // Meter timeSignature; // is it important for Midi ??
86 } _Sequence;
87
88 // Sequence[]
89 typedef struct _Arr_Sequence {
90     size_t len;
91     _Sequence* arr;
92 } _Arr_Sequence;
93
94
95 typedef struct Part {
96     _Arr_Sequence seqs;
97     // Chord keySignature;
98     // Enum Instrument instrument;
99 } Part;
100
101
102 // Part[]
103 typedef struct _Arr_Part {
104     size_t len;
105     Part* arr;
106 } _Arr_Part;
107
108 typedef struct _Score_Singleton {
109     _Arr_Part parts;
110     // Chord keySignature;
111     // int Tempo;
112     // Meter timeSignature = {4, q};
113 } _Score_Singleton;
```

Listing 8.14: stdlib.c

```
/*
 * Authors:
 *  - Rodrigo Manubens
 *  - Ruonan Xu
 */

// clang -emit-llvm -o stdlib.bc -c stdlib.c
// clang -S -emit-llvm -c stdlib.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "beathoven.h"

char _buffer[20];

int _pitch_values[7] = {0,2,4,5,7,9,11};


int len(ptr_t arr_struct_p) {
    return (int)(*((size_t *) arr_struct_p));
}

string _str_of_pitch(pitch p) {
    string _buffer = malloc(4); // garbage!
    char c = '\0';
    if (p->alter == 1) c = '#';
    else if (p->alter == -1) c = 'b';
    sprintf(_buffer, "%c%d%c", p->key, p->octave, c);

    return _buffer;
}

string _str_of_duration(duration d) {
    string _buffer = malloc(10); // garbage!
    sprintf(_buffer, "%d/%d", d->a, d->b);
    return _buffer;
}

string _str_of_Note(Note *note) { // cannot pass the whole struct as parameter
    string _buffer = malloc(14); // garbage!
    pitch p = note->p;
    duration d = note->d;
    if (p->alter == 1)
        sprintf(_buffer, "%c%d#:%d/%d", p->key, p->octave, d->a, d->b);
    else if (p->alter == -1)
        sprintf(_buffer, "%c%d#:%d/%d", p->key, p->octave, d->a, d->b);
    else sprintf(_buffer, "%c%d:%d/%d", p->key, p->octave, d->a, d->b);
    return _buffer;
}

void _write_sequence_midi_text(Seq input_sequence){

  int midi_pitches[input_sequence.len];
  float midi_durations[input_sequence.len];

  FILE *file_pointer;
  char sentenc[1000];
```

```
59    char cwd[1000];
60    if (getcwd(cwd, sizeof(cwd)) != NULL) {
61        fprintf(stdout, "Current working dir: %s\n", cwd);
62        strcat(cwd, "/../bet_midi_library/file_example.txt");
63    }
64     else
65         perror("getcwd() error");
66
67    file_pointer = fopen(cwd,"w");
68
69    if(file_pointer == NULL){
70        printf("Error! \n");
71        exit(1);
72    }
73
74    int i;
75
76    for(i=0; i < input_sequence.len; i++){
77      midi_pitches[i] = _get_midi_pitch(input_sequence.arr[i].p);
78      midi_durations[i] = 4.0 * (input_sequence.arr[i].d->a) / input_sequence.arr[i].d->b;
79    }
80
81    for(i=0; i < input_sequence.len; i++){
82       fprintf(file_pointer, "%d,", midi_pitches[i]);
83    }
84    fprintf(file_pointer, "-1\n");
85
86    for(i=0; i < input_sequence.len; i++){
87       fprintf(file_pointer, "%f,", midi_durations[i]);
88    }
89    fprintf(file_pointer, "-1\n");
90
91    fclose(file_pointer);
92
93 }
94
95 void __make_midi_from_midi_text(){
96   const char * script = "./betmidi.sh";
97   system(script);
98 }
99
100 void render_as_midi(Seq * input_sequence){
101     _write_sequence_midi_text(*input_sequence);
102     _make_midi_from_midi_text();
103 }
104
105 int _get_midi_pitch(pitch p) {
106   int note_number_index = 0;
107
108   if( (int)(p->key) == 'A') note_number_index = 5;
109   else if( (int)(p->key) == 'B') note_number_index = 6;
110   else note_number_index = (int)(p->key) - (int)'C';
111
112
113   int note_number = ((p->octave + 1)*12) + _pitch_values[note_number_index] +
          (p->alter);
114
115   return note_number;
116 }
```

```
117
118 /*
119 extern struct pitch p;
120 extern void f(struct pitch p);
121 */
```

Listing 8.15: betmidi.sh

```
#!/bin/bash

cd ../bet_midi_library/
clang++-3.8 -O3 -Wall -Iinclude -std=c++11 -o betmidi
    src-programs/betmidi.cpp -Llib -lmidifile
./betmidi
```

Listing 8.16: create-examples.sh

```
LLI="lli"
BEAT="./beathoven"
helperPrint=1

Check(){
basename=`echo $1 | sed 's/.*\\///
                                s/.bt//'`
  printf $BEAT
  printf $1
  printf $LLI

  eval $BEAT < $1 > "$basename.ll"
  eval $LLI "$basename.ll"
  cp "../bet_midi_library/twinkle.mid" "../example_outputs/"$basename.mid


}

INPUTS="../example/*.bt"
for file in $INPUTS
do
  # echo "$file"
  Check $file
done
```

8.2 Midi Library Wrapper

Listing 8.17: test_midi.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```
3  #include "beathoven.h"
4
5
6  int main(){
7
8  // testing note ----------------------------------
9
10 _pitch pitch_struct;
11
12 pitch_struct.key = 'C';
13 pitch_struct.octave = 4;
14 pitch_struct.alter = 1;
15
16 _duration duration_struct;
17
18 duration_struct.a = 1;
19 duration_struct.b = 4;
20
21 pitch test_pitch = &pitch_struct;
22 duration test_duration = &duration_struct;
23
24 Note test_note;
25 test_note.p = test_pitch;
26 test_note.d = test_duration;
27
28 printf("Note duration: %.4f \n", ((double)test_note.d->a) / test_note.d->b);
29 printf("Note pitch: %c %d \n", test_note.p->key, test_note.p->octave);
30
31 // test get midi pitch function
32 int final_pitch_number = _get_midi_pitch(test_pitch);
33
34 char str_number_pitch[10];
35 sprintf(str_number_pitch,"%d\n", final_pitch_number);
36 puts(str_number_pitch);
37
38
39 // make other pitch, make into sequence and test with library functions
40 _pitch second_pitch_struct;
41
42 second_pitch_struct.key = 'D';
43 second_pitch_struct.octave = 4;
44 second_pitch_struct.alter = 0;
45
46 _duration second_duration_struct;
47
48 second_duration_struct.a = 1;
49 second_duration_struct.b = 4;
50
51 pitch second_test_pitch = &second_pitch_struct;
52 duration second_test_duration = &second_duration_struct;
53
54 Note second_test_note;
55 second_test_note.p = second_test_pitch;
56 second_test_note.d = second_test_duration;
57
58
59 Seq test_sequence;
60 int len = 2;
61 Note note_arr[2];
```

```
62 note_arr[0] = test_note;
63 note_arr[1] = second_test_note;
64
65 test_sequence.arr = note_arr;
66 test_sequence.len = 2;
67
68 __write_sequence_midi_text(test_sequence);
69 __make_midi_from_midi_text();
70
71 }
```

Listing 8.18: bet_wrapper_main.cpp

```
1 #include <cstdlib>
2 #include <fstream>
3 #include <iostream>
4 #include "call_exe.h"
5
6 int main()
7 {
8     call_exe();
9      // execute the UNIX command "ls -l >test.txt"
10    // std::system("./createmidifile");
11
12 }
```

Listing 8.19: call_exe.cpp

```
1 #include <cstdlib>
2 #include <fstream>
3 #include <iostream>
4 #include "call_exe.h"
5
6 void call_exe()
7 {
8   std::system("clang++-3.8 -O3 -Wall -Iinclude -std=c++11 -o betmidi
        src-programs/betmidi.cpp -Llib -lmidifile");
9   std::system("./betmidi");
10 }
```

Listing 8.20: call_exe.h

```
1 void call_exe();
```