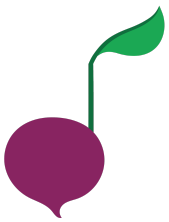


Sick Beets

Final Report

Manager: Courtney Wong (cw2844)
Language Guru: Angel Yang (cy2389)
System Architect: Kevin Shen (ks3206)
Tester: Jin Peng (jjp2172)

Table of Contents

1. Introduction

Sick Beets is a programming language that allows users to compose music by generating MIDI files. Sick Beets is inspired by the structured nature of music, which makes it easy to represent a composition piece by defining attributes of notes such as pitch or duration. Using Sick Beets, users can concatenate and layer series of notes to digitally encode their compositions.

The language revolves around the idea of tunes, which are a series of notes attached to durations. Users can easily put together arrays of notes and durations to create a tune. In addition, users can concatenate tunes together as well as layer them on top of each other. Through a series of built in functions and operators, users are able to set the tempo, use different instruments, and layer their tunes together. Finally, users can render their tunes into MIDI files, which can be played to hear the final product. In order to create the MIDI files, Sick Beets integrates with the C++ Library, CFugue.

2. Tutorial

2.1 Compiling a Program

Steps to run the compiler and execute a program:

1. Compile the CFugue Library and return to the root directory

   ```
   $ cd CFugue
   $ cmake CMakeLists.txt
   $ make
   $ cd ..
   ```

2. Make the compiler

   ```
   $ make
   ```

3. Compile the .sb file into LLVM

   ```
   $ ./sb.native < test_file.sb > test_file_llvm.ll
   ```

4. Run the LLVM code

```
$ lli test_file_llvm.ll
```

2.2 Making Music: Mary had a Little Lamb

Below is a simple program to generate a MIDI file for the song, “Mary had a Little Lamb”:

```
1     /* mary_had_a_lamb.sb */
2
3     tune x = [e,d,c,d,e,e,e,r] : i
4     tune y = [d,d,d,r,e,g,g,r] : i
5     tune z = [e,d,d,e,d,c] : i
6
7     tune mary = x.y.x.z
8     render (mary, "mary_had_a_lamb.midi")
9
10    tune mary_flute = setInstrument(mary, “flute”)
11    render (mary_flute, “mary_flute.midi”)
```

Lines 3-5 create tunes called x,y,z. The duration on the right side of the : operator is applied to each of the notes in the array on the left side of the operator. Line 7 puts the song together. The tunes are concatenated together with the . operator. Line 9 creates a MIDI file from the tune. The default instrument for the tune is the piano. There are a series of standard library functions listed in Section 3.5 to manipulate the tempo and instruments of the tunes.

An example of manipulating a tune is setting the instrument. On Line 10, the instrument flute is applied to the mary tune, and a new tune is returned. Line 11 renders a new file called “mary_flute.midi”.

Once you compile the file as described in 2.1, then the “mary_had_a_lamb.midi” and “mary_flute.midi” files will be created in the root directory.

2.3 Defining Functions

It is also possible to write your own functions. Here is a short program to demonstrate how to write a common function like gcd:

```
1     int gcd(int x, int y) {
2             if(y == 0) {
3                     return x
4             }
5             else {
6                     return gcd(y, x % y)
7             }
8     }
9     print(gcd (10,35))
```

Line 1 specifies that the function will return an int. The name of the function is gcd, and it takes two parameters that are ints, x and y. Line 2 shows an example of control-flow. The if statement first checks if y is equal to 0. If it is, then it proceeds to Line 3. Otherwise, gcd calls itself in Line 6.

The function is then called in Line 9 with arguments 10 and 35. The result of this call is then printed out using the built-in print function.

3. Language Reference Manual

3.1 Types and Literals

3.1.1 Primitive Types

Boolean (bool) : May be true or false.

Integer (int) : A 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$

Floating Point (float) : A single-precision 32-bit IEEE 754 floating point, with a decimal point such as 2.75 or an exponent part such as 1e 5, or both.

String (string) : A sequence of ASCII characters such as “hello world \n!” The string literal is enclosed in quotes, and special characters are escaped using a backslash. The supported escape sequences are:

\n newline	\r carriage return
\t horizontal tab	\v vertical tab

\\ backslash \” double quote

Note: A pitch is specified by the following form: *NoteAccidentalOctave-Offset.* Accidentals are optional and can be specified by: ‘b’ for flats and ‘#’ for sharps. All offsets must be positive, and c5 represents middle C. For example, **eb6** is an E-flat which is one octave above middle C. Supported notes include a,b,c,d,e,f,g and r for rests.

Duration: The duration of a note is specified by a combination of key letters. We support the following durations:

w: whole	i: eighth
h: half	s: sixteenth
q: quarter	t: thirty-second

Letters can be combined together to create other durations: For example, **wh** would be a dotted whole duration.

3.1.2 Arrays

Array literals are literals enclosed by hard brackets. Elements in the array are separated by commas. The following are valid arrays:

```
int[] intArray = [ 1, 2, 3, 4 ]
string[] fruitList = [ “apple”, “orange” ]
```

Arrays are strongly typed, and all arrays can only have items of the same type. For example, [1, w, “red”] is not a valid array.

3.2 Operators and Expressions

3.2.1 Identifiers

Variable and function identifiers are sequences of one or more letters and digits where the first character is a letter. Variable and function identifiers may not be any sequence of characters that could represent a note or duration. Here are several examples of identifiers:

chorus1, printHello, song2

The following are invalid identifiers that result in a syntax error:

```
1train, sick-beets, _hi, a, wh
```

3.2.2 Assignment Operator

The operator = denotes assignment of an expression to a variable identifier. The variable type must be specified at declaration. The expression to the right of the = must evaluate to the type in the declaration. Variables must be assigned to a value at declaration.

Examples of a valid variable assignment:

```
int x = 5
note[] notes = [a, b, c]
```

Examples of invalid variable assignment:

```
int a = 1 // a is reserved as a note
float z = “string” // a float variable cannot be assigned to a
                      duration
```

3.2.3 Arithmetic Operators

The arithmetic operators are +, -, *, and /. These are all left to right associative, with * and / have higher precedence than + and -.

Table 3.2.3 explains what each arithmetic operator does:

Operator	Explanation
+	Adds values of left and right operands.
-	Subtracts value of right operand from value of left operand.
*	Multiplies values of left and right operands.
/	Divides value of left operand by value of right operand.

Table 3.2.3: Explanations of arithmetic operators

Arithmetic operators may be applied to arguments of either float or int type. Both expressions that the operator is being applied to must be of the same type.

3.2.4 Relational and Logical Operators

Below the arithmetic operators in precedence are the relational operators: >, >=, <, and <=. These operators all have the same precedence. Just below the relational operators in precedence are the equality operators: ==, !=. Below the equality operators is boolean AND: &&, and then boolean OR: ||.

Table 3.2 explains the relational and logical operators, ordered in decreasing precedence.

Operators	**Explanation**
>, >=,<, <=	The relational operators. These compare the values of the left and right operands and evaluate as true or false.
==, !=	The equality operators. These determine whether the left and right operands are equal in value or not.
&&	Boolean AND. Expects left and right operands to be of type boolean.
||	Boolean OR. Expects left and right operands to be of type boolean.

Table 3.2.4: Explanations of relational and logical operators

3.2.5 Array Operators

Arrays are accessed with the following syntax:

identifier[index]

The index must have type int, and must range from 0 to (array length - 1). Elements of an array can be modified using the assignment operator, or retrieved. For example:

```
string[] fruits = [ “apple”, “orange” ]
print fruits[1] // prints “orange”
fruits[0] = “banana”
print fruits[2] // syntax error
```

3.2.6 Musical Operators

: augment - The : operator applies notes to rhythms to create a tune. Notes and rhythms can be augmented in a one-to-many relationship.

```
tune = [ g, a, b, c ] : w
tune = d : [ q, q, q, q ]
tune = a : q
```

If array is augmented with another array, each array must have the same number of elements, or an error will be thrown.

```
tune = [ c, e, d ] : [ q, q, h ]
```

3.2.7 Tunes

A tune is a series of notes with corresponding durations for a given instrument. The default instrument for a tune is piano, but a tune can have any instrument.

```
tune = [ c, e, d ] : [ q, q, h ]
violin_tune = setInstrument(tune, “violin”)
flute_tune = setInstrument(tune, “flute”)
```

We can concatenate tunes as well using the . operator.

```
piano_tune = [ c, e, d ] : [ q, q, h ] . [ g ] : [ w ]
```

3.2.8 Phrases

A phrase is a combination of tunes played at the same time. Phrases can be created using the *addLayer* built-in function explained in the standard library functions section.

```
chorus = addLayer( piano_tune, violin_tune, “1” }
```

3.2.9 Comments

Comments are enclosed by /* */.

```
/* This is a single line comment. */

/*
```

```
   This is a
   multi-line
   comment.
*/
```

3.3 Control Flow

3.3.1 if else

Keywords “if” and “else” denote conditional statements in which the expression body associated with each conditional is executed iff the boolean expression evaluates to true:

```
if ( /* boolean expression */ ) {
     /* expression body */
} else {
     /* expression body */
}
```

An if statement can stand alone or be followed by an else block. An else-block cannot appear alone.

3.3.2 for loop

Keyword “for” denotes the for loop, which behaves like the Java for loop. The *initialization* expression executes once and initializes the loop. The loop will continue until the *termination* expression evaluates to *false*. The *increment* expression executes after every iteration of the loop.

```
for ( /* initialization */; /* termination */; /* increment */ ) {
     /* expression body */
}
```

3.3.3 while loop

Keyword “while” denotes the while loop that will execute the expression body repeatedly as long as the boolean expression remains true:

```
while ( /* boolean expression */ ) {
```

```
        /* expression body */
    }
```

3.4 Program Structure

3.4.1 Functions
3.4.1.1 Defining a Function

Keyword “function” denotes the definition of a function:

```
    function_type function_name ( /* list of parameters */ ) {
        /* expression body */
        return /* item or value returned */
    }
```

Following function return type comes the name of the function and (a parameter)* enclosed in parenthesis. The return value of the function must have a type matching the function type. Functions can also be type void, and in that case a return statement is not necessary.

3.4.1.2 Applying a Function

```
    transposed_song = transpose_song (song, 5)
```

Note: When a function is applied, parameters will be constant.

3.4.2 Scope

Variables are statically scoped. The scope of variables is the outermost level of braces in which it is defined. If a variable is declared and not confined with braces, then the scope is the whole program and is considered a global variable. Variables cannot be initialized with the same name as a global variable.

3.5 Standard Library Functions

The standard library contains functions helpful for manipulating tracks. Sick Beets comes with three standard library functions: **print**, **render**, **setInstrument**, **setTempo**, and **addLayer**. Each is outlined below:

print function
Prints the argument to standard out. Any expression that is a basic type can be printed.

```
print (“string”)
```

render function
Takes two arguments. The first is a tune and the second is a string, which will be the name of the outputted MIDI file. Creates a MIDI file.

```
render (song_name, “test_midi.midi”)
```

setInstrument function
Returns a tune. The first argument is a tune, and the second argument is an instrument as a string. The function returns a tune with that instrument applied.

```
tune violin_tune = setInstrument (tune, “violin”)
```

setTempo function
Returns a tune. The first argument is a tune, and the second argument is an integer, which is the tempo to set the tune for. The function returns a tune with that tempo applied. If setTempo is not called, the tempo of the tune is by default 120 bpm.

```
tune fast_tune = setTempo (tune, 300)
```

addLayer function
Returns a tune. The first and second arguments are tunes, and the third argument is a string which is the channel the second tune will be in. There are 16 MIDI channels, named 1-16. The first tune is by default played on channel 0.

```
tune layered = addLayer(tune1, tune2, “1”)
```

3.6 Context-Free Grammar

```
program → decls
 EOF
```

```
decls → epsilon | decls stmt | decls func_decl
stms → epsilon | stmts stmt
stmt → WHILE LPAREN expr RPAREN stmt
      | LBRACE stmts RBRACE EOL
      | RETURN expr EOL
      | IF LPAREN expr RPAREN stmt ELSE stmt
      | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt
      | expr EOL
      | EOL
expr → INTLIT | FLOATLIT | STRINGLIT | TRUE | FALSE | ID
      | NOTELIT | DURATION LIT | TUNELIT
      | expr PLUS expr | expr DOT expr | expr MINUS expr
      | expr COLON expr | expr TIMES expr | expr DIVIDE expr
      | expr MODULUS expr | expr LESS expr | expr GREATER expr
      | expr LESS expr | expr GREATER expr | expr LEQ expr
      | expr GEQ expr | expr EQ expr | expr NEQ
      | expr AND expr | expr OR expr
      | MINUS expr %prec NEG | NOT expr
      | typ ID ASSIGN expr | ID ASSIGN expr | ID ASSIGN expr
      | ID ASSIGN expr | ID LPAREN actuals_opt RPAREN
      | LBRACKET elements_opt RBRACKET
      | primitive LBRACKET expr RBRACKET
      | ID LBRACKET expr RBRACKET
      | ID LBRACKET expr RBRACKET ASSIGN expr
typ → primitive | primitive LBRACKET RBRACKET
primitive → INT | FLOAT | BOOL | STRING | VOID
      | NOTE | DURATION | TUNE | CHAR
actuals_opt → epsilon | actuals_list
actuals_list → expr | actuals_list COMMA expr
func_decl → typ ID LPAREN formals_opt RPAREN LBRACE stmts RBRACE
formals_opt → epsilon | formals_list
formal_list → typ ID | formal_list COMMA typ ID
elements_opt → epsilon | elements_list
elements_list → expr | elements_list COMMA expr
```

4. Project Plan

4.1 Planning Process

Our team worked well together because we met frequently, either two or three times per week. At our first meetings, we set up our project plan and created an outline of what we wanted to accomplish every week for

our project. At first, the milestones that were assigned to us were the project proposal and language reference manual (LRM), but after that we found ourselves having to keep track of our own time and scheduling our own milestones. By having the language reference manual on hand, we were able to better map out which features we wanted to accomplish. The first milestone required that was related to programming was Hello World. This took more time than we had expected, and we ended up meeting frequently in order to finish this by the deadline. After this milestone, we reflected on our initial timeline and changed it based on experience with Hello World. We updated our timeline to more realistically reflect how much work we thought we could get done in one week.

4.2 Development Process

The first tasks were to develop a project proposal and an LRM. After these tasks, the first thing we did was set up a Github repository. Whenever we pushed code, we would let the other teammates know in order to decrease the potential for conflicting code. From those, we were able to easily envision what our scanner, parser, and ast were going to look like, since we included a context free grammar in our original LRM.

Throughout the development process, we added and removed functionalities of our language as we saw fit. Depending on what we thought would be plausible to implement, we changed some functionalities. For example, we altered some functionalities that were originally going to be operators so that they would be built-in functions instead. Once we had a lot of our basic language functionalities down, we were able to move on to the core music functionalities. We faced a lot of trouble when it came to using, CFugue, a C++ library, which is the library that we used to produce the MIDI files. At first we had wanted to be able to directly link our LLVM code with the CFugue library, but we had a lot of trouble linking it because the library's sample code had a complicated CMake file. Based on the advice of our TA, we edited CFugue's sample application and executed it from our LLVM code. We then pass the notes we want to generate as command line arguments to the executable created.

4.3 Testing Process

We created our test suite based on the MicroC test suite, and we would always run our tests before pushing a new feature. At first, our tests were geared towards making sure our scanner and parser were working as expected, and towards the end we were writing tests to make sure that our programs were behaving as expected.

We modified the original MicroC testall.sh to be more informative with a progress bar (in color, much like git!) to show at a glance what tests were passing and not passing. In addition, we would pipe the relevant outputs/error messages directly to stdout when running testall.sh so we did not have to dig through logs and output files to debug our language.

Below is an example run of testall.sh with test errors just to demo its output:

```
dyn-***-**-***-***:sick-beets jin$ ./testall.sh
-n test-arrays...
OK
-n test-arraysAssign...
OK
…

-n fail-reassign...
OK
-n fail-redeclaration...
OK
----------------------------------------------------------------------
++++++++++++++++++++++++++++++++++++++++++++++++++++++ --
52 out of 54 tests passed

Failing Tests:
 test-error
1c1
< 115
\ No newline at end of file
---
> not 115
\ No newline at end of file
 test-error2
Fatal error: exception Parsing.Parse_error
----------------------------------------------------------------------
```

If there is a difference in the outputs, the expected and actual outputs are printed to stdout. Otherwise, the error produced is printed to stdout.

4.4 Programming Style

As a team, we agreed to follow the following guidelines for OCaml programming:

- No lines greater than 80 characters.
- Use tab indentations, and ensure the tab width is 4 spaces.
- Use meaningful names for all files and functions.
- Indent to indicate scope.
- Use comments at the beginning of functions for explanatory purposes.
- Pattern match as much as possible.
- Use a pipe character | to match all cases except the first one.
- Be as specific as possible when throwing errors.
- Refactor code so that there is a minimal amount of repeated code.
- Use lowercase and underscores in naming.
- Try not to put multiple statements on one line.
- Add newlines between separate thoughts in the code.
- Keep related code in the same block of code.

The most important thing was making sure that the code was understandable to the other members of the team. All of the elements of our programming style were meant to make sure that our code was readable to each other so that the other members would be able to resume work that other people had been doing.

4.5 Roles and Responsibilities

Courtney: Project Manager
Kevin: System Architect
Angel: Language Guru
Jin: Tester

Even though we had assigned roles, each member touched almost all of the code. For more information on specific roles, refer to the Section 4.7, Project Log.

4.6 Programming Environment

Our development environment was made up of OCaml, Bash, and LLVM. OCaml was used to write the compiler components, Bash was used to write the test script, and LLVM was used to compile and test the code.

Three members of our teams developed in an OS X environment, and one member used the Ubuntu Virtual Machine that was suggested by the instructors. Some members used Vim to edit files, and some members used Sublime Text. In general, Vim was used to edit and create the test files.

We created a private Github repository for version control, and worked on all of our team documents using Google Docs. All of this helped in creating easy collaboration.

We sustained ourselves with boba from Shiny Tea located on 102nd and Broadway.

4.7 Projected Timeline

2/8 - Project Proposal, assign roles
2/22 - Language reference manual
3/4 - Scanner, parser, ast
3/10 - “hello world” working
3/27 - “hello world” due
4/2 - control flow, variables, function declaration, function calls, booleans
4/9 - connect to CFugue, render a MIDI file
4/16 - Arrays, for loops, augment notes to rhythms, stdlib function - render
4/23 - stdlib functions - layer, settempo, setinstrument, fix up augment

4.6 Project Log

The commits for our project without commits for merging.

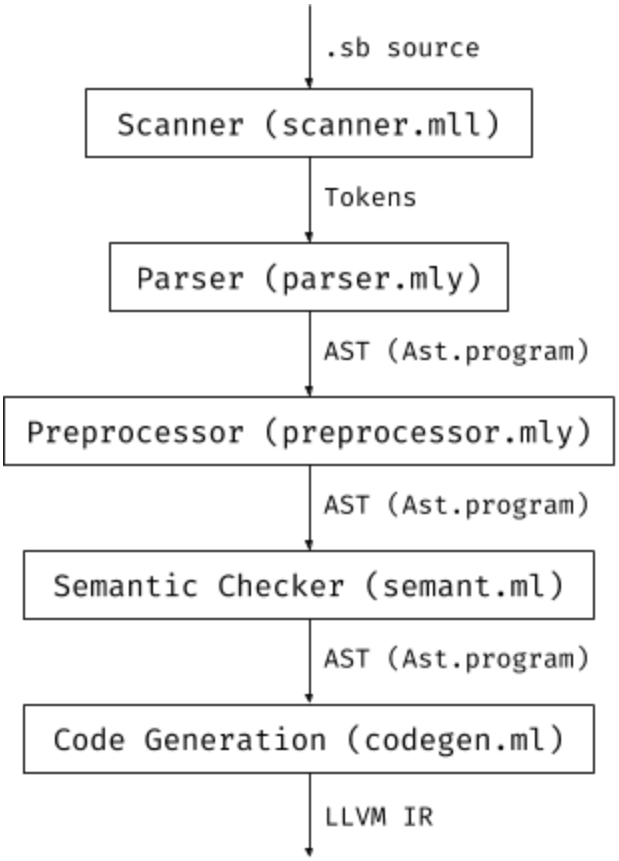
```
2017-05-10 16:00:33 Jin Peng cleaned tests
2017-05-08 21:25:38 courtneywong add demo
2017-05-07 17:50:02 courtneywong clean up some things
2017-05-07 16:11:49 courtneywong small change to codegen
2017-05-07 14:35:42 Kevin Shen Add builder param to build_string function
2017-05-06 22:46:16 Jin Peng added more fail tests for semant
2017-05-06 22:37:08 Kevin Shen Add strcpy call to Augment calls so computer
doesn't print garbage
2017-05-06 21:39:16 Angel readme
2017-05-02 17:02:36 Jin Peng updated testall script
2017-05-02 16:58:54 Jin Peng added fib test
2017-05-01 17:59:12 Angel change module name
2017-05-01 17:53:09 Angel change microc to sick-beets
2017-05-01 17:51:13 courtneywong remove all compiler warnings
2017-05-01 17:18:12 Angel clean up code
2017-05-01 16:39:17 Jin Peng added mod, gcd
2017-04-27 21:29:58 courtneywong add semantic checking for add layer
2017-04-27 21:27:05 Angel add semant for setTempo
2017-04-27 21:24:43 courtneywong layer semant
2017-04-27 21:21:52 Kevin Shen Semantic checking for setInstrument()
2017-04-27 21:11:47 Angel change set tempo from operator to function
2017-04-27 20:54:45 courtneywong add layer
2017-04-27 20:38:51 Angel set tempo function
2017-04-27 18:51:03 Angel tempo operator
2017-04-24 20:12:55 courtneywong change concatenations to copy first
2017-04-24 17:50:40 Angel add set tempo operator
2017-04-24 17:50:19 courtneywong add notes
2017-04-24 17:49:42 Kevin Shen Add instruments test
2017-04-24 17:46:38 Kevin Shen Add Char type to AST and fix strcat issue for
setInstrument
2017-04-22 18:12:19 Kevin Shen Instrument progress
2017-04-22 16:07:54 courtneywong fix warnings
2017-04-22 16:06:22 Kevin Shen Starting instruments
2017-04-22 16:05:41 Angel remove commented code
2017-04-22 15:55:45 courtneywong something
2017-04-22 15:48:18 Angel gitignore
2017-04-22 15:46:25 Angel render: output file name and expr evaluator
2017-04-22 14:42:16 Angel render
2017-04-20 18:57:31 courtneywong add concat tests
2017-04-20 18:53:50 courtneywong enable variables for tunes
2017-04-20 17:15:37 Kevin Shen Edit tests
2017-04-20 16:48:36 courtneywong strcat
2017-04-20 16:48:21 courtneywong try to make strcat work
2017-04-20 16:47:21 Jin Peng added comma for arrays
2017-04-20 16:29:15 Jin Peng parser no comma
2017-04-17 18:00:05 Angel render testing
```

```
2017-04-17 17:58:18 Kevin Shen Fix global variable initialization for arrays
2017-04-17 17:53:02 Jin Peng fixed sr conflict w comma in eelement lsit
2017-04-17 17:52:59 courtneywong fix printing arrays
2017-04-17 17:41:48 Angel  semant check, dynamically render string
2017-04-17 17:12:19 Angel app takes in string of notes and durations to create
MIDI file
2017-04-17 17:01:49 Angel add CFugue files to gitignore
2017-04-17 16:58:13 Angel remove CFugue CMakeFiles and CMakeCache.txt
2017-04-16 19:16:44 courtneywong decrease errors
2017-04-16 19:05:41 Angel render calls executable
2017-04-1 19:05:36 Kevin Shen Array default initialization and array element
assignment
2017-04-16 18:58:11 courtneywong warnings
2017-04-16 18:57:53 Jin Peng removed ops1 ops2
2017-04-16 18:50:11 Jin Peng cleaned up testall to remove files, more error
messages
2017-04-16 18:36:29 courtneywong tunes work
2017-04-16 18:27:23 courtneywong stuff
2017-04-16 18:26:58 Angel build string
2017-04-16 18:25:15 Angel build string
2017-04-16 18:24:54 courtneywong fix global vars
2017-04-16 18:22:42 Jin Peng fixed merge conflicts for EOL,testall update
2017-04-16 18:18:13 Jin Peng fix sr w EOL, update testall
2017-04-16 17:28:44 courtneywong tunes
2017-04-16 17:20:43 courtneywong tune test
2017-04-16 17:20:01 Angel array pointer access
2017-04-15 18:16:31 courtneywong add tune type
2017-04-14 01:25:23 courtneywong add for
2017-04-14 01:01:37 Angel print float/strings
2017-04-14 00:00:00 Kevin Shen New array lit syntax
2017-04-13 23:14:10 courtneywong add notes
2017-04-13 21:13:48 courtneywong add note and duration types
2017-04-13 21:03:16 courtneywong recognize dur and note
2017-04-13 19:52:49 Angel take out printInt, printBool, printFloat, etc. to
just print
2017-04-11 18:17:00 courtneywong fix arrays in semant
2017-04-11 17:38:27 courtneywong arrays in semant
2017-04-11 17:28:45 Kevin Shen Arrays of any type now
2017-04-11 16:29:48 Kevin Shen Array creation and access for int arrays
2017-04-10 17:45:18 courtneywong add float.out
2017-04-10 17:44:19 Angel semant checking for print
2017-04-10 17:43:45 courtneywong add floating operations
2017-04-10 17:17:53 Jin Peng tests for not, negative, bool
2017-04-10 17:16:31 courtneywong add negative
2017-04-06 23:31:27 courtneywong fix while loop
2017-04-06 23:26:52 Angel update schedule
2017-04-06 23:24:22 Angel while semant checking
2017-04-06 23:07:24 Angel test redeclaration
```

```
2017-04-06 22:59:25 Angel check for double declaration
2017-04-06 22:55:49 Jin Peng added test-var.outs, CFugue
2017-04-06 22:51:30 Angel gitignore
2017-04-06 22:48:46 Angel reassign semant checking
2017-04-06 21:27:26 Angel variable semant checking
2017-04-06 21:02:17 courtneywong add cfugue folder
2017-04-06 20:46:49 courtneywong add function void test
2017-04-06 20:46:18 courtneywong readd notes/durations
2017-04-05 20:56:50 courtneywong fix shift/reduce errors
2017-04-05 17:23:48 courtneywong be able to return floats and strings
2017-04-03 22:03:42 Jin Peng updated testall.sh
2017-04-03 21:47:36 Jin Peng fixed test-vars1, vars2, scanner.mll for if/else
2017-04-03 21:46:26 Kevin Shen Add reassign node to AST/codegen, reassign test
2017-04-03 21:12:19 courtneywong fix if then parsing
2017-04-03 20:51:02 Angel semant check for multiple print types
2017-04-03 20:23:11 Angel semantic checking for functions
2017-04-03 20:22:42 courtneywong add output files
2017-04-03 20:17:31 courtneywong add new tests for func decls
2017-04-02 20:56:12 Jin Peng fail func tests
2017-04-02 20:19:15 Kevin Shen Local variables
2017-04-02 20:16:35 courtneywong return ints and bools
2017-04-02 18:52:47 courtneywong add more function types
2017-03-30 23:21:40 Jin Peng fixed test-funcDecl
2017-03-30 23:20:38 Jin Peng updated color progress bar
2017-03-30 23:00:58 courtneywong add bracket to funcdecl test
2017-03-30 22:59:37 Jin Peng added fail-func tests, color progress bar
2017-03-30 22:56:55 Jin Peng added tests for fail-func, color progress bar
2017-03-30 22:55:12 courtneywong add comments to codegen
2017-03-30 22:25:23 Angel semant checker, add return to functions
2017-03-30 22:18:46 Kevin Shen Fix tests after adding preprocessor
2017-03-30 22:10:19 Kevin Shen Preprocessor to add statements into a main
function
2017-03-30 22:08:34 courtneywong add id to expr
2017-03-30 21:43:38 courtneywong fix func decl test
2017-03-30 21:27:25 courtneywong function decls and calls to codegen
2017-03-30 21:23:36 Jin Peng fixed parsing err for test-funcDecl
2017-03-30 21:18:19 Angel semant check for print decl or duplicate formals
2017-03-30 21:10:35 courtneywong try to get fdecls to work
2017-03-30 21:05:11 Kevin Shen Variable declarations
2017-03-30 20:52:54 Angel test function
2017-03-30 20:40:56 Jin Peng funcDecl test added
2017-03-29 11:53:54 courtneywong hack to fix if stmts
2017-03-28 19:02:57 Angel function decl
2017-03-27 20:56:20 courtneywong add if to codegen
2017-03-27 20:47:20 courtneywong add if stmt to scanner/parser
2017-03-27 17:32:13 courtneywong update to do list
2017-03-25 23:16:02 Jin Peng test-out updated for printInt
2017-03-25 23:14:16 courtneywong add todo list
```

```
2017-03-25 22:59:38 courtneywong add tests
2017-03-25 22:54:39 courtneywong print integers/rename tests
2017-03-25 22:39:43 courtneywong print int
2017-03-25 22:39:19 Angel rename tests
2017-03-25 22:19:30 Angel strings with escape characters
2017-03-25 22:16:41 courtneywong reverse statement order
2017-03-25 21:49:18 courtneywong fix strings
2017-03-25 21:23:18 courtneywong fix test-print
2017-03-23 21:07:42 Angel tests multiline print
2017-03-23 21:04:28 Jin Peng update make, testall for sb
2017-03-23 20:56:57 courtneywong add git ignore
2017-03-23 00:16:14 Jin Peng dded basic tests, updated testall.sh for .sb
files and Makefile to clean test outputs
2017-03-21 23:23:18 courtneywong fix shift/reduce errors
2017-03-21 22:23:32 Angel delete ast.mli for ast.ml
2017-03-21 22:18:53 Angel HELLO WORLD WOW ZOMG YAY EXCLAMATIONMARK
2017-03-02 21:01:53 courtneywong add codegen
2017-03-02 20:45:46 courtneywong add microc semantic checker
2017-03-02 20:37:37 courtneywong recognize function call
2017-02-27 18:40:42 Angel music arrays
2017-02-27 17:50:06 kevinshen While loops
2017-02-27 17:43:40 kevinshen While loops
2017-02-27 17:29:33 Jin Peng updated ast with functions + types
2017-02-27 17:27:48 Jin Peng updated ast with function
2017-02-27 17:26:19 courtneywong add type decl to assignment
2017-02-27 16:41:11 courtneywong change
2017-02-27 16:36:31 kevinshen Move assignment to stmt instead of expr
2017-02-26 21:35:19 courtneywong add duration type
2017-02-26 21:30:15 courtneywong add note type
2017-02-26 21:20:40 courtneywong fix boolean
2017-02-25 20:06:27 Jin Peng added ID, ASSIGN
2017-02-25 19:06:42 Jin Peng updated w variables based off microC
2017-02-25 17:34:19 courtneywong resolve conflicts
2017-02-25 17:33:57 courtneywong use digit for float regex
2017-02-25 17:33:26 kevinshen remove redundant stuff
2017-02-25 17:25:35 Angel fix bool printing
2017-02-25 17:23:46 courtneywong combine eval float and int
2017-02-25 17:14:43 courtneywong add floats
2017-02-25 17:14:06 Angel Bool as bool
2017-02-25 17:13:13 Angel Bool as bool
2017-02-25 17:05:11 courtneywong floats
2017-02-25 17:04:57 kevinshen Pretty print with parentheses
2017-02-25 16:54:13 courtneywong add && and ||
2017-02-25 16:49:17 courtneywong add strings
2017-02-25 16:29:40 courtneywong precendence
2017-02-25 16:22:00 courtneywong add booleans
2017-02-25 15:30:07 Angel starter code for scanner, parser, ast
2017-02-06 17:37:41 Angel Initial commit
```

5. Architecture



The front-end analysis of the Sick-beets compiler consists of lexical analysis and parsing of tokens into an abstract syntax tree (AST). This is followed by preprocessing of the AST, static semantic analysis, and code generation into LLVM IR. To render MIDI files, an LLVM instruction executes a C++ program that uses the CFugue library.

5.1 Scanner (scanner.mll) - Angel, Jin

The scanner performs lexical analysis on the source file and generates a stream of tokens to simplify the job of the parser.

scanner.mll consists of a set a regular expressions with associated semantic actions. ocamllex takes the scanner.mll file as input and outputs a lexical analyzer written in OCaml. This lexical analyzer then

takes a .sb source file and generates tokens, which include keywords, identifiers, operators, and literals.

The scanner ignores whitespace, tabs, and carriage returns. Comments are implemented by ignoring all characters between an opening “/*” token and a closing “*/” token. The scanner also converts escape sequences (e.g. \”) in string literals.

5.2 Parser (parser.mly) - Courtney, Angel, Jin

The parser takes the stream of tokens generated by the scanner and constructs an abstract syntax tree (AST) using the context-free grammar defined in parser.mly. ocamlyacc is used to create the parser from the grammar specified in parser.mly.

5.2.1 Abstract Syntax Tree (ast.ml) - Courtney, Angel, Jin

The abstract syntax tree data structure is defined in ast.ml. The root node is a program, and the program is a tuple of (lists of statements, lists of functions).

5.3 Pre-processor (preprocessor.ml) - Kevin

The pre-processor organizes all statements in the statement list into a main function, so that the AST is converted from a tuple of (statement list, function list) to a function list. This allows the semantic checker and code generator to simply iterate through all function definitions.

5.4 Semantic Checker (semant.ml) - Kevin, Courtney, Angel

The semantic checker traverses through the modified AST to ensure that the program is semantically correct. It makes sure that functions and variables are not declared multiple times, and that no user-defined functions have the same name as any reserved function. Types of variables are checked in actual arguments of functions, assignment of variables, return values of functions, and expressions with binary operators. When a variable name is used, the semantic checker first looks for the variable in the local variable map, and then in the global variable map.

If no errors are thrown, then the AST is passed to codegen.ml for code generation. The code generator is guaranteed to generate LLVM code without throwing any errors.

5.5 Code Generator (codegen.ml) - Kevin, Courtney, Angel

The code generator traverses the semantically checked AST and generates LLVM IR. The function definition for the main function is constructed first, and all variables initialized in the main function are added to a global variable map. This way, all other functions have access to and can modify these global variables. Variables initialized in these other functions are added to their own local variable map.

In order to render MIDI files, we compiled a C++ program that uses CFugue as a statically linked library. The resulting executable takes a string of notes and an output file name as arguments, and renders the notes using a CFugue library function. When the render() reserved function is called in a .sb program, the compiler creates an LLVM instruction that runs the executable.

6. Testing Plan

6.1 Sample Programs

6.1.1 tests/test-fib.sb

```
int fib(int n) {
        if(n > 1) {
                return fib(n - 1) + fib(n - 2)
        }
        else {
                return n
        }
}

note[] notes = [a,b,c,d,e,f,g]
int n = 1
tune fibonacci = a:i
while(n <= 10) {
        note result = notes[fib(n)%7]
        tune x = result : i
        fibonacci = fibonacci.x
        n = n + 1
```

```
 18 }
 19
 20 render(fibonacci, "fib.midi")
```

```
./sb.native < examples/fib.sb > fib.ll
vim fib.ll
```

```
 1 ; ModuleID = 'sick-beets'
  2
  3 @notes = global i8** null
  4 @note = private unnamed_addr constant [2 x i8] c"a\00"
  5 @note.1 = private unnamed_addr constant [2 x i8] c"b\00"
  6 @note.2 = private unnamed_addr constant [2 x i8] c"c\00"
  7 @note.3 = private unnamed_addr constant [2 x i8] c"d\00"
  8 @note.4 = private unnamed_addr constant [2 x i8] c"e\00"
  9 @note.5 = private unnamed_addr constant [2 x i8] c"f\00"
 10 @note.6 = private unnamed_addr constant [2 x i8] c"g\00"
 11 @n = global i32 0
 12 @tune = private unnamed_addr constant [1 x i8] zeroinitializer
 13 @fibonacci = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune, i
32 0, i32 0)
 14 @string = private unnamed_addr constant [2 x i8] c" \00"
 15 @string.7 = private unnamed_addr constant [1 x i8] zeroinitializer
 16 @note.8 = private unnamed_addr constant [2 x i8] c"a\00"
 17 @duration = private unnamed_addr constant [2 x i8] c"i\00"
 18 @string.9 = private unnamed_addr constant [2 x i8] c" \00"
 19 @note.10 = private unnamed_addr constant [1 x i8] zeroinitializer
 20 @result = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @note.10, i
32 0, i32 0)
 21 @tune.11 = private unnamed_addr constant [1 x i8] zeroinitializer
 22 @x = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.11, i32 0,
i32 0)
 23 @string.12 = private unnamed_addr constant [2 x i8] c" \00"
 24 @string.13 = private unnamed_addr constant [1 x i8] zeroinitializer
 25 @duration.14 = private unnamed_addr constant [2 x i8] c"i\00"
 26 @string.15 = private unnamed_addr constant [2 x i8] c" \00"
 27 @string.16 = private unnamed_addr constant [27 x i8] c"CFugue/bin/testCFugue
Lib \22\00"
 28 @string.17 = private unnamed_addr constant [3 x i8] c"\22 \00"
 29 @string.18 = private unnamed_addr constant [9 x i8] c"fib.midi\00"
 30
 31 declare i32 @printf(i8*, ...)
 32
 33 declare i8* @strcat(i8*, i8*)
 34
 35 declare i8* @strcpy(i8*, i8*)
 36
 37 define i32 @main() {
```

```
 38 entry:
 39   %malloccall = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoin
t (i1** getelementptr (i1*, i1** null, i32 1) to i32), i32 7), i32 1), i32 p
trtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32)))
 40   %tmp = bitcast i8* %malloccall to i8**
 41   %tmp1 = getelementptr i8*, i8** %tmp, i32 1
 42   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note, i32 0, i32 0)
, i8** %tmp1
 43   %tmp2 = getelementptr i8*, i8** %tmp, i32 2
 44   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.1, i32 0, i32
0), i8** %tmp2
 45   %tmp3 = getelementptr i8*, i8** %tmp, i32 3
 46   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.2, i32 0, i32
0), i8** %tmp3
 47   %tmp4 = getelementptr i8*, i8** %tmp, i32 4
 48   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.3, i32 0, i32
0), i8** %tmp4
 49   %tmp5 = getelementptr i8*, i8** %tmp, i32 5
 50   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.4, i32 0, i32
0), i8** %tmp5
 51   %tmp6 = getelementptr i8*, i8** %tmp, i32 6
 52   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.5, i32 0, i32
0), i8** %tmp6
 53   %tmp7 = getelementptr i8*, i8** %tmp, i32 7
 54   store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @note.6, i32 0, i32
0), i8** %tmp7
 55   store i8** %tmp, i8*** @notes
 56   store i32 1, i32* @n
 57   %tmp9 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint (i8*
getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32 ptrtoin    t
(i8* getelementptr (i8, i8* null, i32 1) to i32)))
 58   %tmp110 = call i8* @strcpy(i8* %tmp9, i8* getelementptr inbounds ([2 x i8]
, [2 x i8]* @note.8, i32 0, i32 0))
 59   %tmp211 = call i8* @strcat(i8* %tmp9, i8* getelementptr inbounds ([2 x i8]
, [2 x i8]* @duration, i32 0, i32 0))
 60   %strcat = call i8* @strcat(i8* %tmp9, i8* getelementptr inbounds ([2 x i8]
, [2 x i8]* @string.9, i32 0, i32 0))
 61   store i8* %strcat, i8** @fibonacci
 62   br label %while
 63
 64 while:                                           ; preds = %while_body, %en
try
 65   %n28 = load i32, i32* @n
 66   %tmp29 = icmp sle i32 %n28, 10
 67   br i1 %tmp29, label %while_body, label %merge
 68
 69 while_body:                                      ; preds = %while
 70   %n = load i32, i32* @n
```

```
 71   %fib_result = call i32 @fib(i32 %n)
 72   %tmp12 = urem i32 %fib_result, 7
 73   %tmp13 = add i32 %tmp12, 1
 74   %tmp14 = load i8**, i8*** @notes
 75   %tmp15 = getelementptr i8*, i8** %tmp14, i32 %tmp13
 76   %tmp16 = load i8*, i8** %tmp15
 77   store i8* %tmp16, i8** @result
 78   %result = load i8*, i8** @result
 79   %tmp18 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint (i8
* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32 ptrtoi
nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
 80   %tmp19 = call i8* @strcpy(i8* %tmp18, i8* %result)
 81   %strcat20 = call i8* @strcat(i8* %tmp19, i8* getelementptr inbounds ([2 x
i8], [2 x i8]* @duration.14, i32 0, i32 0))
 82   store i8* %strcat20, i8** @x
 83   %fibonacci = load i8*, i8** @fibonacci
 84   %x = load i8*, i8** @x
 85   %tmp22 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint (i8
* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32 ptrtoi
nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
 86   %tmp123 = call i8* @strcpy(i8* %tmp22, i8* %fibonacci)
 87   %tmp224 = call i8* @strcat(i8* %tmp123, i8* %x)
 88   %strcat25 = call i8* @strcat(i8* %tmp224, i8* getelementptr inbounds ([2 x
i8], [2 x i8]* @string.15, i32 0, i32 0))
 89   store i8* %strcat25, i8** @fibonacci
 90   %n26 = load i32, i32* @n
 91   %tmp27 = add i32 %n26, 1
 92   store i32 %tmp27, i32* @n
 93   br label %while
 94
 95 merge:                                            ; preds = %while
 96   %fibonacci30 = load i8*, i8** @fibonacci
 97   %tmp32 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint (i8
* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32 ptrtoi
nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
 98   %strcpy = call i8* @strcpy(i8* %tmp32, i8* getelementptr inbounds ([27 x i
8], [27 x i8]* @string.16, i32 0, i32 0))
 99   %strcat33 = call i8* @strcat(i8* %strcpy, i8* %fibonacci30)
100   %strcat34 = call i8* @strcat(i8* %strcat33, i8* getelementptr inbounds ([3
x i8], [3 x i8]* @string.17, i32 0, i32 0))
101   %strcat35 = call i8* @strcat(i8* %strcat34, i8* getelementptr inbounds ([9
x i8], [9 x i8]* @string.18, i32 0, i32 0))
102   %" " = getelementptr inbounds i8, i8* %strcat35, i32 0
103   %rendf = call i32 @system(i8* %" ")
104   ret i32 0
105 }
106
107 define i32 @fib(i32 %n) {
```

```
entry:
  %n1 = alloca i32
  store i32 %n, i32* %n1
  %n2 = load i32, i32* %n1
  %tmp = icmp sgt i32 %n2, 1
  br i1 %tmp, label %then, label %else

merge:                                            ; No predecessors!
  ret i32 0

then:                                             ; preds = %entry
  %n3 = load i32, i32* %n1
  %tmp4 = sub i32 %n3, 1
  %fib_result = call i32 @fib(i32 %tmp4)
  %n5 = load i32, i32* %n1
  %tmp6 = sub i32 %n5, 2
  %fib_result7 = call i32 @fib(i32 %tmp6)
  %tmp8 = add i32 %fib_result, %fib_result7
  ret i32 %tmp8

else:                                             ; preds = %entry
  %n9 = load i32, i32* %n1
  ret i32 %n9
}

declare i32 @system(i8*)

declare noalias i8* @malloc(i32)
```

6.1.2 tests/test-rickroll.sb

```
tune romanticize(tune song) {
        tune newSong = setTempo(song, 80)
        newSong = setInstrument(newSong, "alto_sax")
        return newSong
}

tune rick = [f,g,bb,g] : s
tune roll1r = [d6,d6,c6] : [is, is, qi]
tune roll2r = [c6,c6,bb] : [is, is, qi]
tune roll3r = [bb,c6,a,g,f,f,c6,bb] : [q,i,is,s,q,i,q,h]

tune rest = r : q
tune left1 = [e4,e4,f4] : [is, is, qi]
tune left2 = [d4,d4,g4] : [is, is, qi]
tune left3 = [c4,g4,f4,d4,g4] : [qi, qi, q, q,h]
```

```
16
17 tune rickRoll = rick.roll1r.rick.roll2r.rick.roll3r
18 tune rickRollLeft = rest.left1.rest.left2.rest.left3
19
20 rickRoll = addLayer(rickRoll, rickRollLeft, "1")
21 render(rickRoll,"rickroll.midi")
22
23 tune rickRollFast = setTempo(rickRoll, 200)
24 render(rickRollFast, "rickroll_fast.midi")
25
26 tune romantic = romanticize(rickRoll)
27 render(romantic, "rickroll_romantic.midi")
```

```
./sb.native < examples/test-rickroll.sb > rickroll.ll
vim rickroll.ll
```

```
  1 ; ModuleID = 'sick-beets'
  2
  3 @tune = private unnamed_addr constant [1 x i8] zeroinitializer
  4 @rick = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune, i32
0,     i32 0)
  5 @string = private unnamed_addr constant [2 x i8] c" \00"
  6 @string.1 = private unnamed_addr constant [1 x i8] zeroinitializer
  7 @duration = private unnamed_addr constant [2 x i8] c"s\00"
  8 @note = private unnamed_addr constant [2 x i8] c"f\00"
  9 @note.2 = private unnamed_addr constant [2 x i8] c"g\00"
 10 @note.3 = private unnamed_addr constant [3 x i8] c"bb\00"
 11 @note.4 = private unnamed_addr constant [2 x i8] c"g\00"
 12 @tune.5 = private unnamed_addr constant [1 x i8] zeroinitializer
 13 @roll1r = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.5,
i3    2 0, i32 0)
 14 @string.6 = private unnamed_addr constant [2 x i8] c" \00"
 15 @string.7 = private unnamed_addr constant [1 x i8] zeroinitializer
 16 @note.8 = private unnamed_addr constant [3 x i8] c"d6\00"
 17 @duration.9 = private unnamed_addr constant [3 x i8] c"is\00"
 18 @note.10 = private unnamed_addr constant [3 x i8] c"d6\00"
 19 @duration.11 = private unnamed_addr constant [3 x i8] c"is\00"
 20 @note.12 = private unnamed_addr constant [3 x i8] c"c6\00"
 21 @duration.13 = private unnamed_addr constant [3 x i8] c"qi\00"
 22 @tune.14 = private unnamed_addr constant [1 x i8] zeroinitializer
 23 @roll2r = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.14,
i    32 0, i32 0)
 24 @string.15 = private unnamed_addr constant [2 x i8] c" \00"
 25 @string.16 = private unnamed_addr constant [1 x i8] zeroinitializer
 26 @note.17 = private unnamed_addr constant [3 x i8] c"c6\00"
 27 @duration.18 = private unnamed_addr constant [3 x i8] c"is\00"
 28 @note.19 = private unnamed_addr constant [3 x i8] c"c6\00"
 29 @duration.20 = private unnamed_addr constant [3 x i8] c"is\00"
```

```
@note.21 = private unnamed_addr constant [3 x i8] c"bb\00"
@duration.22 = private unnamed_addr constant [3 x i8] c"qi\00"
@tune.23 = private unnamed_addr constant [1 x i8] zeroinitializer
@roll3r = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.23,
i    32 0, i32 0)
@string.24 = private unnamed_addr constant [2 x i8] c" \00"
@string.25 = private unnamed_addr constant [1 x i8] zeroinitializer
@note.26 = private unnamed_addr constant [3 x i8] c"bb\00"
@duration.27 = private unnamed_addr constant [2 x i8] c"q\00"
@note.28 = private unnamed_addr constant [3 x i8] c"c6\00"
@duration.29 = private unnamed_addr constant [2 x i8] c"i\00"
@note.30 = private unnamed_addr constant [2 x i8] c"a\00"
@duration.31 = private unnamed_addr constant [3 x i8] c"is\00"
@note.32 = private unnamed_addr constant [2 x i8] c"g\00"
@duration.33 = private unnamed_addr constant [2 x i8] c"s\00"
@note.34 = private unnamed_addr constant [2 x i8] c"f\00"
@duration.35 = private unnamed_addr constant [2 x i8] c"q\00"
@note.36 = private unnamed_addr constant [2 x i8] c"f\00"
@duration.37 = private unnamed_addr constant [2 x i8] c"i\00"
@note.38 = private unnamed_addr constant [3 x i8] c"c6\00"
@duration.39 = private unnamed_addr constant [2 x i8] c"q\00"
@note.40 = private unnamed_addr constant [3 x i8] c"bb\00"
@duration.41 = private unnamed_addr constant [2 x i8] c"h\00"
@tune.42 = private unnamed_addr constant [1 x i8] zeroinitializer
@rest = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.42,
i32     0, i32 0)
@string.43 = private unnamed_addr constant [2 x i8] c" \00"
@string.44 = private unnamed_addr constant [1 x i8] zeroinitializer
@note.45 = private unnamed_addr constant [2 x i8] c"r\00"
@duration.46 = private unnamed_addr constant [2 x i8] c"q\00"
@string.47 = private unnamed_addr constant [2 x i8] c" \00"
@tune.48 = private unnamed_addr constant [1 x i8] zeroinitializer
@left1 = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.48,
i3    2 0, i32 0)
@string.49 = private unnamed_addr constant [2 x i8] c" \00"
@string.50 = private unnamed_addr constant [1 x i8] zeroinitializer
@note.51 = private unnamed_addr constant [3 x i8] c"e4\00"
@duration.52 = private unnamed_addr constant [3 x i8] c"is\00"
@note.53 = private unnamed_addr constant [3 x i8] c"e4\00"
@duration.54 = private unnamed_addr constant [3 x i8] c"is\00"
@note.55 = private unnamed_addr constant [3 x i8] c"f4\00"
@duration.56 = private unnamed_addr constant [3 x i8] c"qi\00"
@tune.57 = private unnamed_addr constant [1 x i8] zeroinitializer
@left2 = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.57,
i3    2 0, i32 0)
@string.58 = private unnamed_addr constant [2 x i8] c" \00"
@string.59 = private unnamed_addr constant [1 x i8] zeroinitializer
@note.60 = private unnamed_addr constant [3 x i8] c"d4\00"
```

```
 74 @duration.61 = private unnamed_addr constant [3 x i8] c"is\00"
 75 @note.62 = private unnamed_addr constant [3 x i8] c"d4\00"
 76 @duration.63 = private unnamed_addr constant [3 x i8] c"is\00"
 77 @note.64 = private unnamed_addr constant [3 x i8] c"g4\00"
 78 @duration.65 = private unnamed_addr constant [3 x i8] c"qi\00"
 79 @tune.66 = private unnamed_addr constant [1 x i8] zeroinitializer
 80 @left3 = global i8* getelementptr inbounds ([1 x i8], [1 x i8]* @tune.66,
i3    2 0, i32 0)
 81 @string.67 = private unnamed_addr constant [2 x i8] c" \00"
 82 @string.68 = private unnamed_addr constant [1 x i8] zeroinitializer
 83 @note.69 = private unnamed_addr constant [3 x i8] c"c4\00"
 84 @duration.70 = private unnamed_addr constant [3 x i8] c"qi\00"
 85 @note.71 = private unnamed_addr constant [3 x i8] c"g4\00"
 86 @duration.72 = private unnamed_addr constant [3 x i8] c"qi\00"
 87 @note.73 = private unnamed_addr constant [3 x i8] c"f4\00"
 88 @duration.74 = private unnamed_addr constant [2 x i8] c"q\00"
 89 @note.75 = private unnamed_addr constant [3 x i8] c"d4\00"
 90 @duration.76 = private unnamed_addr constant [2 x i8] c"q\00"
 91 @note.77 = private unnamed_addr constant [3 x i8] c"g4\00"
 92 @duration.78 = private unnamed_addr constant [2 x i8] c"h\00"
 93 @tune.79 = private unnamed_addr constant [1 x i8] zeroinitializer
 94 @rickRoll = global i8* getelementptr inbounds ([1 x i8], [1 x i8]*
@tune.79,     i32 0, i32 0)
 95 @string.80 = private unnamed_addr constant [2 x i8] c" \00"
 96 @string.81 = private unnamed_addr constant [2 x i8] c" \00"
 97 @string.82 = private unnamed_addr constant [2 x i8] c" \00"
 98 @string.83 = private unnamed_addr constant [2 x i8] c" \00"
 99 @string.84 = private unnamed_addr constant [2 x i8] c" \00"
100 @tune.85 = private unnamed_addr constant [1 x i8] zeroinitializer
101 @rickRollLeft = global i8* getelementptr inbounds ([1 x i8], [1 x i8]*
@tune    .85, i32 0, i32 0)
102 @string.86 = private unnamed_addr constant [2 x i8] c" \00"
103 @string.87 = private unnamed_addr constant [2 x i8] c" \00"
104 @string.88 = private unnamed_addr constant [2 x i8] c" \00"
105 @string.89 = private unnamed_addr constant [2 x i8] c" \00"
106 @string.90 = private unnamed_addr constant [2 x i8] c" \00"
107 @string.91 = private unnamed_addr constant [2 x i8] c"1\00"
108 @string.92 = private unnamed_addr constant [2 x i8] c"V\00"
109 @string.93 = private unnamed_addr constant [2 x i8] c" \00"
110 @string.94 = private unnamed_addr constant [27 x i8]
c"CFugue/bin/testCFugue    Lib \22\00"
111 @string.95 = private unnamed_addr constant [3 x i8] c"\22 \00"
112 @string.96 = private unnamed_addr constant [14 x i8] c"rickroll.midi\00"
113 @tune.97 = private unnamed_addr constant [1 x i8] zeroinitializer
114 @rickRollFast = global i8* getelementptr inbounds ([1 x i8], [1 x i8]*
@tune    .97, i32 0, i32 0)
115 @string.98 = private unnamed_addr constant [4 x i8] c"200\00"
116 @string.99 = private unnamed_addr constant [2 x i8] c" \00"
```

```
117 @string.100 = private unnamed_addr constant [2 x i8] c"T\00"
118 @string.101 = private unnamed_addr constant [27 x i8]
c"CFugue/bin/testCFugu    eLib \22\00"
119 @string.102 = private unnamed_addr constant [3 x i8] c"\22 \00"
120 @string.103 = private unnamed_addr constant [19 x i8]
c"rickroll_fast.midi\0    0"
121 @tune.104 = private unnamed_addr constant [1 x i8] zeroinitializer
122 @romantic = global i8* getelementptr inbounds ([1 x i8], [1 x i8]*
@tune.104    , i32 0, i32 0)
123 @string.105 = private unnamed_addr constant [27 x i8]
c"CFugue/bin/testCFugu    eLib \22\00"
124 @string.106 = private unnamed_addr constant [3 x i8] c"\22 \00"
125 @string.107 = private unnamed_addr constant [23 x i8]
c"rickroll_romantic.mi    di\00"
126 @string.108 = private unnamed_addr constant [3 x i8] c"80\00"
127 @string.109 = private unnamed_addr constant [2 x i8] c" \00"
128 @string.110 = private unnamed_addr constant [2 x i8] c"T\00"
129 @string.111 = private unnamed_addr constant [9 x i8] c"alto_sax\00"
130 @string.112 = private unnamed_addr constant [3 x i8] c"I[\00"
131 @string.113 = private unnamed_addr constant [3 x i8] c"] \00"
132
133 declare i32 @printf(i8*, ...)
134
135 declare i8* @strcat(i8*, i8*)
136
137 declare i8* @strcpy(i8*, i8*)
138
139 define i32 @main() {
140 entry:
141   %tmp = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8*     getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoint     (i8* getelementptr (i8, i8* null, i32 1) to i32)))
142   %tmp1 = call i8* @strcpy(i8* %tmp, i8* getelementptr inbounds ([1 x i8],
[    1 x i8]* @string.1, i32 0, i32 0))
143   %tmp2 = call i8* @strcat(i8* %tmp1, i8* getelementptr inbounds ([2 x
i8],     [2 x i8]* @note, i32 0, i32 0))
144   %tmp3 = call i8* @strcat(i8* %tmp1, i8* getelementptr inbounds ([2 x
i8],     [2 x i8]* @duration, i32 0, i32 0))
145   %strcat = call i8* @strcat(i8* %tmp1, i8* getelementptr inbounds ([2 x
i8]    , [2 x i8]* @string, i32 0, i32 0))
146   %tmp22 = call i8* @strcat(i8* %strcat, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @note.2, i32 0, i32 0))
147   %tmp33 = call i8* @strcat(i8* %strcat, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @duration, i32 0, i32 0))
148   %strcat4 = call i8* @strcat(i8* %strcat, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string, i32 0, i32 0))
149   %tmp25 = call i8* @strcat(i8* %strcat4, i8* getelementptr inbounds ([3 x
i    8], [3 x i8]* @note.3, i32 0, i32 0))
```

```
150   %tmp36 = call i8* @strcat(i8* %strcat4, i8* getelementptr inbounds ([2 x
i    8], [2 x i8]* @duration, i32 0, i32 0))
151   %strcat7 = call i8* @strcat(i8* %strcat4, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string, i32 0, i32 0))
152   %tmp28 = call i8* @strcat(i8* %strcat7, i8* getelementptr inbounds ([2 x
i    8], [2 x i8]* @note.4, i32 0, i32 0))
153   %tmp39 = call i8* @strcat(i8* %strcat7, i8* getelementptr inbounds ([2 x
i    8], [2 x i8]* @duration, i32 0, i32 0))
154   %strcat10 = call i8* @strcat(i8* %strcat7, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string, i32 0, i32 0))
155   %strcat11 = call i8* @strcat(i8* %tmp1, i8* getelementptr inbounds ([2 x
i    8], [2 x i8]* @string, i32 0, i32 0))
156   store i8* %strcat11, i8** @rick
157   %tmp12 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
158   %tmp13 = call i8* @strcpy(i8* %tmp12, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.7, i32 0, i32 0))
159   %tmp214 = call i8* @strcat(i8* %tmp13, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.8, i32 0, i32 0))
160   %tmp315 = call i8* @strcat(i8* %tmp13, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @duration.9, i32 0, i32 0))
161   %strcat16 = call i8* @strcat(i8* %tmp13, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.6, i32 0, i32 0))
162   %tmp217 = call i8* @strcat(i8* %strcat16, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.10, i32 0, i32 0))
163   %tmp318 = call i8* @strcat(i8* %strcat16, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.11, i32 0, i32 0))
164   %strcat19 = call i8* @strcat(i8* %strcat16, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.6, i32 0, i32 0))
165   %tmp220 = call i8* @strcat(i8* %strcat19, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.12, i32 0, i32 0))
166   %tmp321 = call i8* @strcat(i8* %strcat19, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.13, i32 0, i32 0))
167   %strcat22 = call i8* @strcat(i8* %strcat19, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.6, i32 0, i32 0))
168   %strcat23 = call i8* @strcat(i8* %tmp13, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.6, i32 0, i32 0))
169   store i8* %strcat23, i8** @roll1r
170   %tmp24 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
171   %tmp26 = call i8* @strcpy(i8* %tmp24, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.16, i32 0, i32 0))
172   %tmp227 = call i8* @strcat(i8* %tmp26, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.17, i32 0, i32 0))
173   %tmp328 = call i8* @strcat(i8* %tmp26, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @duration.18, i32 0, i32 0))
```

```
174   %strcat29 = call i8* @strcat(i8* %tmp26, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.15, i32 0, i32 0))
175   %tmp230 = call i8* @strcat(i8* %strcat29, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.19, i32 0, i32 0))
176   %tmp331 = call i8* @strcat(i8* %strcat29, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.20, i32 0, i32 0))
177   %strcat32 = call i8* @strcat(i8* %strcat29, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.15, i32 0, i32 0))
178   %tmp233 = call i8* @strcat(i8* %strcat32, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.21, i32 0, i32 0))
179   %tmp334 = call i8* @strcat(i8* %strcat32, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.22, i32 0, i32 0))
180   %strcat35 = call i8* @strcat(i8* %strcat32, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.15, i32 0, i32 0))
181   %strcat36 = call i8* @strcat(i8* %tmp26, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.15, i32 0, i32 0))
182   store i8* %strcat36, i8** @roll2r
183   %tmp37 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
184   %tmp38 = call i8* @strcpy(i8* %tmp37, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.25, i32 0, i32 0))
185   %tmp239 = call i8* @strcat(i8* %tmp38, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.26, i32 0, i32 0))
186   %tmp340 = call i8* @strcat(i8* %tmp38, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @duration.27, i32 0, i32 0))
187   %strcat41 = call i8* @strcat(i8* %tmp38, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.24, i32 0, i32 0))
188   %tmp242 = call i8* @strcat(i8* %strcat41, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.28, i32 0, i32 0))
189   %tmp343 = call i8* @strcat(i8* %strcat41, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.29, i32 0, i32 0))
190   %strcat44 = call i8* @strcat(i8* %strcat41, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
191   %tmp245 = call i8* @strcat(i8* %strcat44, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @note.30, i32 0, i32 0))
192   %tmp346 = call i8* @strcat(i8* %strcat44, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.31, i32 0, i32 0))
193   %strcat47 = call i8* @strcat(i8* %strcat44, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
194   %tmp248 = call i8* @strcat(i8* %strcat47, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @note.32, i32 0, i32 0))
195   %tmp349 = call i8* @strcat(i8* %strcat47, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.33, i32 0, i32 0))
196   %strcat50 = call i8* @strcat(i8* %strcat47, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
197   %tmp251 = call i8* @strcat(i8* %strcat50, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @note.34, i32 0, i32 0))
```

```
198   %tmp352 = call i8* @strcat(i8* %strcat50, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.35, i32 0, i32 0))
199   %strcat53 = call i8* @strcat(i8* %strcat50, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
200   %tmp254 = call i8* @strcat(i8* %strcat53, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @note.36, i32 0, i32 0))
201   %tmp355 = call i8* @strcat(i8* %strcat53, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.37, i32 0, i32 0))
202   %strcat56 = call i8* @strcat(i8* %strcat53, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
203   %tmp257 = call i8* @strcat(i8* %strcat56, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.38, i32 0, i32 0))
204   %tmp358 = call i8* @strcat(i8* %strcat56, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.39, i32 0, i32 0))
205   %strcat59 = call i8* @strcat(i8* %strcat56, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
206   %tmp260 = call i8* @strcat(i8* %strcat59, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.40, i32 0, i32 0))
207   %tmp361 = call i8* @strcat(i8* %strcat59, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @duration.41, i32 0, i32 0))
208   %strcat62 = call i8* @strcat(i8* %strcat59, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.24, i32 0, i32 0))
209   %strcat63 = call i8* @strcat(i8* %tmp38, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.24, i32 0, i32 0))
210   store i8* %strcat63, i8** @roll3r
211   %tmp64 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
212   %tmp165 = call i8* @strcpy(i8* %tmp64, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @note.45, i32 0, i32 0))
213   %tmp266 = call i8* @strcat(i8* %tmp64, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @duration.46, i32 0, i32 0))
214   %strcat67 = call i8* @strcat(i8* %tmp64, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.47, i32 0, i32 0))
215   store i8* %strcat67, i8** @rest
216   %tmp68 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
217   %tmp69 = call i8* @strcpy(i8* %tmp68, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.50, i32 0, i32 0))
218   %tmp270 = call i8* @strcat(i8* %tmp69, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.51, i32 0, i32 0))
219   %tmp371 = call i8* @strcat(i8* %tmp69, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @duration.52, i32 0, i32 0))
220   %strcat72 = call i8* @strcat(i8* %tmp69, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.49, i32 0, i32 0))
221   %tmp273 = call i8* @strcat(i8* %strcat72, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.53, i32 0, i32 0))
```

```
222   %tmp374 = call i8* @strcat(i8* %strcat72, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.54, i32 0, i32 0))
223   %strcat75 = call i8* @strcat(i8* %strcat72, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.49, i32 0, i32 0))
224   %tmp276 = call i8* @strcat(i8* %strcat75, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.55, i32 0, i32 0))
225   %tmp377 = call i8* @strcat(i8* %strcat75, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.56, i32 0, i32 0))
226   %strcat78 = call i8* @strcat(i8* %strcat75, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.49, i32 0, i32 0))
227   %strcat79 = call i8* @strcat(i8* %tmp69, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.49, i32 0, i32 0))
228   store i8* %strcat79, i8** @left1
229   %tmp80 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
230   %tmp81 = call i8* @strcpy(i8* %tmp80, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.59, i32 0, i32 0))
231   %tmp282 = call i8* @strcat(i8* %tmp81, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.60, i32 0, i32 0))
232   %tmp383 = call i8* @strcat(i8* %tmp81, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @duration.61, i32 0, i32 0))
233   %strcat84 = call i8* @strcat(i8* %tmp81, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.58, i32 0, i32 0))
234   %tmp285 = call i8* @strcat(i8* %strcat84, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.62, i32 0, i32 0))
235   %tmp386 = call i8* @strcat(i8* %strcat84, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.63, i32 0, i32 0))
236   %strcat87 = call i8* @strcat(i8* %strcat84, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.58, i32 0, i32 0))
237   %tmp288 = call i8* @strcat(i8* %strcat87, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.64, i32 0, i32 0))
238   %tmp389 = call i8* @strcat(i8* %strcat87, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.65, i32 0, i32 0))
239   %strcat90 = call i8* @strcat(i8* %strcat87, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.58, i32 0, i32 0))
240   %strcat91 = call i8* @strcat(i8* %tmp81, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.58, i32 0, i32 0))
241   store i8* %strcat91, i8** @left2
242   %tmp92 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i8    * getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrtoi    nt (i8* getelementptr (i8, i8* null, i32 1) to i32)))
243   %tmp93 = call i8* @strcpy(i8* %tmp92, i8* getelementptr inbounds ([1 x
i8]    , [1 x i8]* @string.68, i32 0, i32 0))
244   %tmp294 = call i8* @strcat(i8* %tmp93, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @note.69, i32 0, i32 0))
245   %tmp395 = call i8* @strcat(i8* %tmp93, i8* getelementptr inbounds ([3 x
i8    ], [3 x i8]* @duration.70, i32 0, i32 0))
```

```
246   %strcat96 = call i8* @strcat(i8* %tmp93, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.67, i32 0, i32 0))
247   %tmp297 = call i8* @strcat(i8* %strcat96, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @note.71, i32 0, i32 0))
248   %tmp398 = call i8* @strcat(i8* %strcat96, i8* getelementptr inbounds ([3
x     i8], [3 x i8]* @duration.72, i32 0, i32 0))
249   %strcat99 = call i8* @strcat(i8* %strcat96, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.67, i32 0, i32 0))
250   %tmp2100 = call i8* @strcat(i8* %strcat99, i8* getelementptr inbounds
([3     x i8], [3 x i8]* @note.73, i32 0, i32 0))
251   %tmp3101 = call i8* @strcat(i8* %strcat99, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @duration.74, i32 0, i32 0))
252   %strcat102 = call i8* @strcat(i8* %strcat99, i8* getelementptr inbounds
([    2 x i8], [2 x i8]* @string.67, i32 0, i32 0))
253   %tmp2103 = call i8* @strcat(i8* %strcat102, i8* getelementptr inbounds
([3     x i8], [3 x i8]* @note.75, i32 0, i32 0))
254   %tmp3104 = call i8* @strcat(i8* %strcat102, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @duration.76, i32 0, i32 0))
255   %strcat105 = call i8* @strcat(i8* %strcat102, i8* getelementptr inbounds
(    [2 x i8], [2 x i8]* @string.67, i32 0, i32 0))
256   %tmp2106 = call i8* @strcat(i8* %strcat105, i8* getelementptr inbounds
([3     x i8], [3 x i8]* @note.77, i32 0, i32 0))
257   %tmp3107 = call i8* @strcat(i8* %strcat105, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @duration.78, i32 0, i32 0))
258   %strcat108 = call i8* @strcat(i8* %strcat105, i8* getelementptr inbounds
(    [2 x i8], [2 x i8]* @string.67, i32 0, i32 0))
259   %strcat109 = call i8* @strcat(i8* %tmp93, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.67, i32 0, i32 0))
260   store i8* %strcat109, i8** @left3
261   %rick = load i8*, i8** @rick
262   %roll1r = load i8*, i8** @roll1r
263   %tmp110 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
264   %tmp1111 = call i8* @strcpy(i8* %tmp110, i8* %rick)
265   %tmp2112 = call i8* @strcat(i8* %tmp1111, i8* %roll1r)
266   %strcat113 = call i8* @strcat(i8* %tmp2112, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.80, i32 0, i32 0))
267   %rick114 = load i8*, i8** @rick
268   %tmp115 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
269   %tmp1116 = call i8* @strcpy(i8* %tmp115, i8* %strcat113)
270   %tmp2117 = call i8* @strcat(i8* %tmp1116, i8* %rick114)
271   %strcat118 = call i8* @strcat(i8* %tmp2117, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.81, i32 0, i32 0))
272   %roll2r = load i8*, i8** @roll2r
```

```
273   %tmp119 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
274   %tmp1120 = call i8* @strcpy(i8* %tmp119, i8* %strcat118)
275   %tmp2121 = call i8* @strcat(i8* %tmp1120, i8* %roll2r)
276   %strcat122 = call i8* @strcat(i8* %tmp2121, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.82, i32 0, i32 0))
277   %rick123 = load i8*, i8** @rick
278   %tmp124 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
279   %tmp1125 = call i8* @strcpy(i8* %tmp124, i8* %strcat122)
280   %tmp2126 = call i8* @strcat(i8* %tmp1125, i8* %rick123)
281   %strcat127 = call i8* @strcat(i8* %tmp2126, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.83, i32 0, i32 0))
282   %roll3r = load i8*, i8** @roll3r
283   %tmp128 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
284   %tmp1129 = call i8* @strcpy(i8* %tmp128, i8* %strcat127)
285   %tmp2130 = call i8* @strcat(i8* %tmp1129, i8* %roll3r)
286   %strcat131 = call i8* @strcat(i8* %tmp2130, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.84, i32 0, i32 0))
287   store i8* %strcat131, i8** @rickRoll
288   %rest = load i8*, i8** @rest
289   %left1 = load i8*, i8** @left1
290   %tmp132 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
291   %tmp1133 = call i8* @strcpy(i8* %tmp132, i8* %rest)
292   %tmp2134 = call i8* @strcat(i8* %tmp1133, i8* %left1)
293   %strcat135 = call i8* @strcat(i8* %tmp2134, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.86, i32 0, i32 0))
294   %rest136 = load i8*, i8** @rest
295   %tmp137 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
296   %tmp1138 = call i8* @strcpy(i8* %tmp137, i8* %strcat135)
297   %tmp2139 = call i8* @strcat(i8* %tmp1138, i8* %rest136)
298   %strcat140 = call i8* @strcat(i8* %tmp2139, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.87, i32 0, i32 0))
299   %left2 = load i8*, i8** @left2
300   %tmp141 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
301   %tmp1142 = call i8* @strcpy(i8* %tmp141, i8* %strcat140)
302   %tmp2143 = call i8* @strcat(i8* %tmp1142, i8* %left2)
```

```
303   %strcat144 = call i8* @strcat(i8* %tmp2143, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.88, i32 0, i32 0))
304   %rest145 = load i8*, i8** @rest
305   %tmp146 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
306   %tmp1147 = call i8* @strcpy(i8* %tmp146, i8* %strcat144)
307   %tmp2148 = call i8* @strcat(i8* %tmp1147, i8* %rest145)
308   %strcat149 = call i8* @strcat(i8* %tmp2148, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.89, i32 0, i32 0))
309   %left3 = load i8*, i8** @left3
310   %tmp150 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
311   %tmp1151 = call i8* @strcpy(i8* %tmp150, i8* %strcat149)
312   %tmp2152 = call i8* @strcat(i8* %tmp1151, i8* %left3)
313   %strcat153 = call i8* @strcat(i8* %tmp2152, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.90, i32 0, i32 0))
314   store i8* %strcat153, i8** @rickRollLeft
315   %rickRoll = load i8*, i8** @rickRoll
316   %rickRollLeft = load i8*, i8** @rickRollLeft
317   %tmp154 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
318   %tmp1155 = call i8* @strcpy(i8* %tmp154, i8* %rickRoll)
319   %tmp2156 = call i8* @strcat(i8* %tmp1155, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.92, i32 0, i32 0))
320   %tmp3157 = call i8* @strcat(i8* %tmp2156, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.91, i32 0, i32 0))
321   %tmp4 = call i8* @strcat(i8* %tmp3157, i8* getelementptr inbounds ([2 x
i8    ], [2 x i8]* @string.93, i32 0, i32 0))
322   %strcat158 = call i8* @strcat(i8* %tmp4, i8* %rickRollLeft)
323   store i8* %strcat158, i8** @rickRoll
324   %rickRoll159 = load i8*, i8** @rickRoll
325   %tmp160 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
326   %strcpy = call i8* @strcpy(i8* %tmp160, i8* getelementptr inbounds ([27
x     i8], [27 x i8]* @string.94, i32 0, i32 0))
327   %strcat161 = call i8* @strcat(i8* %strcpy, i8* %rickRoll159)
328   %strcat162 = call i8* @strcat(i8* %strcat161, i8* getelementptr inbounds
(    [3 x i8], [3 x i8]* @string.95, i32 0, i32 0))
329   %strcat163 = call i8* @strcat(i8* %strcat162, i8* getelementptr inbounds
(    [14 x i8], [14 x i8]* @string.96, i32 0, i32 0))
330   %" " = getelementptr inbounds i8, i8* %strcat163, i32 0
331   %rendf = call i32 @system(i8* %" ")
332   %rickRoll164 = load i8*, i8** @rickRoll
```

```
333   %tmp166 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
334   %partial = call i8* @strcpy(i8* %tmp166, i8* getelementptr inbounds ([2
x     i8], [2 x i8]* @string.100, i32 0, i32 0))
335   %partial_2 = call i8* @strcat(i8* %tmp166, i8* getelementptr inbounds
([4     x i8], [4 x i8]* @string.98, i32 0, i32 0))
336   %partial_3 = call i8* @strcat(i8* %tmp166, i8* getelementptr inbounds
([2     x i8], [2 x i8]* @string.99, i32 0, i32 0))
337   %strcat167 = call i8* @strcat(i8* %tmp166, i8* %rickRoll164)
338   store i8* %strcat167, i8** @rickRollFast
339   %rickRollFast = load i8*, i8** @rickRollFast
340   %tmp168 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
341   %strcpy169 = call i8* @strcpy(i8* %tmp168, i8* getelementptr inbounds
([27     x i8], [27 x i8]* @string.101, i32 0, i32 0))
342   %strcat170 = call i8* @strcat(i8* %strcpy169, i8* %rickRollFast)
343   %strcat171 = call i8* @strcat(i8* %strcat170, i8* getelementptr inbounds
(    [3 x i8], [3 x i8]* @string.102, i32 0, i32 0))
344   %strcat172 = call i8* @strcat(i8* %strcat171, i8* getelementptr inbounds
(    [19 x i8], [19 x i8]* @string.103, i32 0, i32 0))
345   %" 173" = getelementptr inbounds i8, i8* %strcat172, i32 0
346   %rendf174 = call i32 @system(i8* %" 173")
347   %rickRoll175 = load i8*, i8** @rickRoll
348   %romanticize_result = call i8* @romanticize(i8* %rickRoll175)
349   store i8* %romanticize_result, i8** @romantic
350   %romantic = load i8*, i8** @romantic
351   %tmp176 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i    8* getelementptr (i8, i8* null, i32 1) to i32), i32 4096), i32 1), i32
ptrto    int (i8* getelementptr (i8, i8* null, i32 1) to i32)))
352   %strcpy177 = call i8* @strcpy(i8* %tmp176, i8* getelementptr inbounds
([27     x i8], [27 x i8]* @string.105, i32 0, i32 0))
353   %strcat178 = call i8* @strcat(i8* %strcpy177, i8* %romantic)
```

6.2 Tests

test-while.sb

```
1 int z = 5
2 while (z < 10) {
3     print("hello")
```

```
 4      z = 10
 5 }
```

test-var2.sb

```
 1 int x = 43
 2 int y = x + 72
 3 print(y)
```

test-var1.sb

```
 1 int variable = 42
 2 print(variable)
```

test-setTempo.sb

```
 1 tune newTune = setTempo ([a,b,c]:[q,w,q], 130)
 2 render ( newTune, "test.midi" )
```

test-rickroll.sb

```
 1 int fib(int n) {
 2         if(n > 1) {
 3                 return fib(n - 1) + fib(n - 2)
 4         }
 5         else {
 6                 return n
 7         }
 8 }
 9
10 note[] notes = [a,b,c,d,e,f,g]
11 int n = 1
12 tune fibonacci = a:i
13 while(n <= 10) {
14         note result = notes[fib(n)%7]
15         tune x = result : i
16         fibonacci = fibonacci.x
17         n = n + 1
18 }
19
20 tune romanticize(tune song) {
21         tune newSong = setTempo(song, 80)
22         newSong = setInstrument(newSong, "alto_sax")
23         return newSong
24 }
25
```

```
render(fibonacci, "fib.midi")

tune rick = [f,g,bb,g] : s
tune roll1r = [d6,d6,c6] : [is, is, qi]
tune roll2r = [c6,c6,bb] : [is, is, qi]
tune roll3r = [bb,c6,a,g,f,f,c6,bb] : [q,i,is,s,q,i,q,h]

tune rest = r : q
tune left1 = [e4,e4,f4] : [is, is, qi]
tune left2 = [d4,d4,g4] : [is, is, qi]
tune left3 = [c4,g4,f4,d4,g4] : [qi, qi, q, q,h]

tune rickRoll = rick.roll1r.rick.roll2r.rick.roll3r
tune rickRollLeft = rest.left1.rest.left2.rest.left3

rickRoll = addLayer(rickRoll, rickRollLeft, "1")
render(rickRoll,"rickroll.midi")

tune rickRollFast = setTempo(rickRoll, 200)
render(rickRollFast, "rickroll_fast.midi")

tune romantic = romanticize(rickRoll)
render(romantic, "rickroll_romantic.midi")
```

test-render.sb

```
render("F6i F6i Rq F6i F6i Rq F6i F6i F6i F6i F6q G6q E6q E6q E6i D6i E6q
E6i D6i     E6i Ai Aq.", "MIDIOutput.midi")
```

test-reassign.sb

```
int n = 42

int foo(int x) {
    x = x + 1
    int y = x * 2
    n = n + y
    return n
}

print(foo(10))
```

test-printMultiLine.sb

```
print("hello\n")
print("hello world")
```

test-printInt.sb

```
1 print(1)
```

test-print.sb

```
1 print("Hello World")
```

test-ops.sb

```
1 print(1 + 2)
2 print("\n")
3 print(1 - 2)
4 print("\n")
5 print(1 * 2)
6 print("\n")
7 print(100 / 2)
8 print("\n")
```

test-not.sb

```
1 bool x = true
2 bool y = 3 < 1
3 print(!true)
4 print("\n")
5 print(!false)
6 print("\n")
7 print(!x)
8 print("\n")
9 print(!y)
```

test-negative.sb

```
1 print(-10)
2 print("\n")
3 print(--42)
```

test-mary.sb

```
1 tune x = [e,d,c,d,e,e,e,r] : i
2 tune y = [d,d,d,r,e,g,g,r] : i
3 tune z = [e,d,d,e,d,c] : i
4
5 tune mary = x.y.x.z
6 tune maryflute = setInstrument(mary, "flute")
7 tune marypiano = setInstrument(mary,"piano")
8
9 render(marypiano.maryflute , "mary.midi")
```

test-localVars.sb

```
 1 int n = 1
 2
 3 int foo(int x, int y) {
 4     return n + x + y
 5 }
 6
 7 print(foo(2, 3))
```

test-layers.sb

```
 1 tune x = [a,b, c] : i
 2 tune y = [d,e,f] : i
 3 tune z = addLayer(x,y,"1")
 4 render(z, "layers.midi")
```

test-instrument.sb

```
 1 /* mary had a little lamb */
 2 tune x = [e, d, c, d, e, e, e] : [q, q, q, q, q, q, q]
 3 tune y = setInstrument(x, "flute")
 4 print(y)
```

test-ifelse.sb

```
 1 if(true) {
 2         print(1)
 3 }
 4 else {
 5         print(2)
 6 }
```

test-hallelujah.sb

```
 1 tune v1r1 = g : [i,q, i, q, i] . a : [i,i,qi] . e : i . g : [i,i,q,i,i] .
a : [q,    i,i]
 2 tune v1r2 = g:[i,i] . a:[i,qi,i,i,q]. [g,g,f,g,g,g] : [i,i,i,i,i,hi]
 3 tune rest = r:[qi]
 4 tune v1r3 = g : [i,q,i,q,i] . a : [q,i] . [b,g] : [q, i] . c6 : [i,i,qi] .
[a,c6,    c6] : i . [d6,r,c6] : [q, i,i]
 5 tune v1r4 = d6 : [i,i,qi,i] . [e6, e6, e6, d6, d6, c6,c6] :
[q,q,i,i,i,i,hq]
 6
 7 tune rest2 = r : i
 8 tune chorusr1 = [e,g,a,a] : [q,i,qi,hi]
 9 tune chorusr2 = [a,g,e,e] : [q,i,qi,hi]
10 tune chorusr3 = [e,g,a,a] : [q,i,qi,hi]
```

```
 11 tune chorusr4 = [a,g,e,f,e,d,c,c] : [q,i,h,i,i,hi,i,hq]
 12
 13 tune chorusl = [g3, f3, a3, a3, f3, f3, c3, g3, c3] : hq
 14
 15 tune right =
v1r1.v1r2.rest.v1r3.v1r4.rest2.chorusr1.rest2.chorusr2.rest2.chorus
r3.rest2.chorusr4
 16
 17 tune l1 = [b3,c3,b3,a3,b3,c3,b3,a3,f3,g3,c3,r,r] : [i, hi,
i,hi,i,hi,i,hq,hq,hq,h    ,i,i]
 18 tune l2 = [d3, c3,f3, g3] : [i, hq, qi, qi] . [a3, f3, g3, e3, a3, a3] :
hq
 19 tune left = l1.rest.l2.chorusl
 20 tune hallelujah = addLayer(right,left,"1")
 21
 22 render(hallelujah, "hallelujah.midi")
```

test-gcd.sb

```
  1 int gcd(int x, int y) {
  2         if(y == 0) {
  3                 return x
  4         }
  5         else {
  6                 return gcd(y, x % y)
  7         }
  8 }
  9 print(gcd(35,20))
 10 print("\n")
 11 print(gcd(5, 24))
```

test-funcDeclVoid.sb

```
  1 void voidFunc(string n) {
  2         print(n)
  3 }
  4 voidFunc("hello")
```

test-funcDeclTune.sb

```
  1 tune funcTune(tune y) {
  2         return y
  3 }
  4 tune x1 = a:w
  5 tune x2 = a : [q, w]
  6 tune x3 = [a, b] : q
  7 tune x4 = [a, b] : [q, w]
  8
```

```
 9 print(funcTune(x1))
10 print(funcTune(x2))
11 print(funcTune(x3))
12 print(funcTune(x4))
```

test-funcDeclStr.sb

```
1 string hello(string n) {
2         return n
3 }
4 print(hello("hello world"))
```

test-funcDeclNote.sb

```
1 note noteFunc(note x) {
2         return x
3 }
4 print(noteFunc(a#9))
5 print(noteFunc(cb))
```

test-funcDeclInt.sb

```
1 int hello(int n) {
2         return n
3 }
4 print(hello(1))
```

test-funcDeclFlt.sb

```
1 float floatFunc(float x) {
2         return x
3 }
4 print(floatFunc(1.1))
```

test-funcDeclDur.sb

```
1 duration durFunc(duration x) {
2         return x
3 }
4 print(durFunc(wh))
```

test-funcDeclBool.s

```
1 bool hello(bool foo) {
2         return foo
3 }
4 print(hello(true))
```

test-for.sb

```
for (int x = 0; x < 5; x = x + 1){
    print("hello")
}
print("\n")
int sum = 0
for (int j = 0; j < 10; j = j + 1) {
    sum = sum + j
}
print(sum)
```

test-float.sb

```
float hello(float n) {
        return n + n
}
print(hello(1.0))
```

test-fib.sb

```
int fib(int n) {
        if(n > 1) {
                return fib(n - 1) + fib(n - 2)
        }
        else {
                return n
        }
}

int n = 0
while(n <= 10) {
        print(fib(n))
        print("\n")
        n = n + 1
}
```

test-concat.sb

```
tune x = [a,b, c] : [q, w, h]
tune y = [a, b, c] : [q, w, w]
tune z = [a,b,c] : [q,w,w] . [a,b,c] : [q,w,h] . [a,b,c] : [q,q,q]
tune z1 = x.y.x
print(z)
print("\n")
print(z1)
```

test-comment.sb

```
 1 /* this is a comment */
 2
 3 int x = 43
 4 /* x = 50 */
 5 int y = x + 72
 6
 7 /*git c
 8 print(y)
 9 print(y)
10 */
11
12 print(y)
```

test-bool.sb

```
 1 print(1 == 2)
 2 print(1 == 1)
 3 print(1 != 2)
 4 print(1 != 1)
 5 print(1 < 2)
 6 print(2 < 1)
 7 print(1 <= 2)
 8 print(1 <= 1)
 9 print(2 <= 1)
10 print(1 > 2)
11 print(2 > 1)
12 print(1 >= 2)
13 print(1 >= 1)
14 print(2 >= 1)
15 print(1 == 2)
16 print(1 == 1)
17 print(1 != 2)
18 print(1 != 1)
19 print(1 < 2)
20 print(2 < 1)
21 print(1 <= 2)
22 print(1 <= 1)
23 print(2 <= 1)
24 print(1 > 2)
25 print(2 > 1)
26 print(1 >= 2)
27 print(1 >= 1)
28 print(2 >= 1)
```

test-augment.sb

```
1 tune x = a:w
```

```
 2 tune y = c:[q,q,q]
 3 print(a : w)
 4 print(c : [q, q, q])
 5 print([c, a, d] : q)
 6 print([c, a, d] : [q, w, h])
```

test-assign.sb

```
 1 int n = 1337
 2 print(n)
 3 print("\n")
 4 int m = 2017
 5 print(m)
```

test-arraysAssign.sb

```
 1 void foo() {
 2     int[] arr1 = int[5]
 3     arr1[2] = 1337
 4     print(arr1[2*2-2])
 5 }
 6 foo()
```

test-arrays.sb

```
 1 void foo() {
 2     int[] arr1 = [1, 2, 3]
 3     print(arr1[2])
 4     print("\n")
 5
 6     bool[] arr2 = [true, true, true, false, true]
 7     print(arr2[3])
 8     print("\n")
 9
10     string[] arr3 = ["hey", "watsup", "hello"]
11     print(arr3[1])
12     print("\n")
13
14     float[] arr4 = [0.1, 3.14, 2.178, 0.25]
15     print(arr4[0])
16     print("\n")
17 }
18
19 foo()
20
21 int[] arr5 = [100, 101]
22 print(arr5[1])
```

fail-redeclaration.sb

```
1 int n = 1
2 int n = 2
```

fail-reassign.sb

```
1 int n = 3
2 n = "string"
```

fail-invalidUnary.sb

```
1 string one = "1"
2 print(-one)
```

fail-illegalUnary.sb

```
1 string one = "1"
2 print(-one)
```

fail-illegalConcat.sb

```
1 note[] notes = [a, b, c]
2 tune tune_1 = [a,b,c] : [q,w,w]
3 tune z =  tune_1 . notes
```

fail-illegalBinary.sb

```
1 int n = 1
2 string str = "one"
3 print(n + str)
```

fail-illegalAugment.sb

```
1 tune x = w : a
```

fail-funcUnrecog.sb

```
1 int n = 10
2 nonexistent_function(10)
```

fail-funcUndecld.sb

```
1 int n = k + 3
```

fail-funcPrint.sb

```
1 int print() {
```

```
  2         return 1
  3 }
```

fail-funcDuplicateFunc.sb

```
  1 int foo() {}
  2 string foo() {}
```

fail-funcDuplicateFormals.sb

```
  1 int foo(string n, int n) {
  2   int var = 1
  3 }
```

fail-assign.sb

```
  1 int n = "string"
```

fail-arrayAccess.sb

```
  1 int[] arr5 = [100, 101]
  2 string one = "1"
  3 print(arr5[one])
```

7. Lessons Learned

7.1 Courtney

This project was definitely one of the most time consuming classes that I have ever taken at Columbia. I learned a lot about compilers, OCaml, and all the problems that you can encounter when compiling to LLVM. It gave me a new appreciation for the languages we use on a regular basis, because there is thought that goes into each and every aspect of the language. One of the hardest things was just making sure we were not forgetting about edge cases when making our code work. While it was pretty simple to get basic functionality working, testing frequently with a variety of programs was extremely important, and often led to us finding bugs in our compiler.

I also learned was that it is really important to keep ourselves accountable for the work that we were doing. Creating a schedule really helped us make sure that we were on track so that we weren't scrambling in the last week right before finals to finish the entire language. Despite it being a lot of work, our team always made sure to have fun.

We celebrated our successes, but also worked together when we were having trouble.

7.2 Angel

Team and Topic: PLT was definitely one of the biggest time-consuming classes I've taken, but I had a pretty cool team and an interesting project topic so I enjoyed working on our project! Make sure you choose a project topic you would be excited to work on for a couple hours every week and a group that you'd be happy to work with.

Reading and Understanding Code: One of the biggest challenges for the project was just tackling such a big code base. The best way to do this is just read a lot of code - our group went through a couple past projects, specifically some of the past music-related projects. Since the code base is so large, I think focusing on one specific feature implemented in the compiler and following its logistic flow through the different parts.

Trade-off between Aesthetics and Compiler Complexity: A lot of what we changed from the original LRM was language design decisions motivated primarily by the aesthetics of the code. We found that some of those decisions would created significantly more complex code for the compiler (i.e. not using commas between elements in an array).

How Compilers Work: At first it was hard to wrap my head around all the different parts of the compiler. However, once I implemented a couple of features, I got a better idea of how it worked and I have to say that it's pretty cool. PLT was hard, but worth it!

7.3 Kevin

This course made me appreciate all of the hard work put into building the languages we use today, since otherwise, we'd still be stuck writing machine code. PLT was also my first exposure to functional programming, and I actually really enjoyed using OCaml. In terms of the project, I learned how important it was for everyone to be on the same page. Almost all of our work was done while we met in person, and this allowed us to ask each other for help whenever. We found pair programming to be useful - sometimes, we even displayed our code on the TV for the entire group to collaborate. Since creating a compiler in a semester isn't a trivial task, staying focused was extremely important. We created weekly checklists and always asked our TA if we were on

track. Lastly, it's possible to make meetings fun and still get work done! We ordered bubble tea to our weekly meetings to make them a little less stressful.

7.4 Jin

Chinese philosopher Laozi once said, “A journey of a thousand miles begins with a single step.” While our language was the biggest technical project I have undertaken at Columbia, it is also the project I started the earliest on and met the most often for. I think having a consistent schedule and making small amounts of progress constantly (even if you are just introducing more bugs or “wasting” a few hours struggling to understand errors) really made a difference in the long run. I learned that making project meetings a weekly commitment and trying to be as productive as possible for the few hours you work on PLT each week saves a lot of stress later down the road. Also, not hesitating to reach out for help and communicating with our TA really gave us opportunities to check our progress and have someone point us in the right direction. Overall, I think beyond the PLT concepts learned in the course, my biggest takeway from this project was learning about the type of work and collaboration needed for a large-scale project.

7.5 Advice to Future Teams

Motivate Each Other: Find a way to keep everyone excited to come to meetings - for our group, we bought a lot of boba throughout the semester. A happy group will be a more productive group!

Track Differences from Original LRM: There will inevitably be changes from the original LRM, but when you are updating your LRM, it can be hard to remember exactly what you need to change. Logging these changes can help make this easier!

Timeline: In the beginning of the semester, our group sat down and wrote out all the features that we wanted to implement. Then, we broke the features down week by week with an extra 2 week buffer at the end. This helped us have a good idea of our progress and stay on track.

Teamwork makes the Dream Work: All of our members worked on all parts of the code, and it was definitely helpful for helping each other debug because we all knew what each part functioned and what the code should have accomplished.

GitHub: This makes keeping track of code changes and maintaining one code base amongst the entire team easy! Plus, knowing GitHub will help with other CS team projects, classes, and maybe even jobs.

Make Meetings Routine: In the beginning of the semester, we would try and find available time between the four of us meeting to meeting. However, it would be hard to find a common time. Then, we set aside one (and later two) times a week to have meetings that made our meetings routine and easier to plan around.

8. Appendix

8.1 Code Listing

8.1.1 .gitignore

```
.DS_Store
tests/.DS_Store
_build/
*.diff
*.ll
*.log
sb.native
CFugue/CMakeCache.txt
CFugue/CMakeFiles/
CFugue/bin/
CFugue/lib/
CFugue/test/CMakeFiles/testCFugueLib.dir/StaticLibTestApp/
CFugue/Makefile
CFugue/cmake_install.cmake
CFugue/test/CMakeFiles/CMakeDirectoryInformation.cmake
CFugue/test/CMakeFiles/testCFugueLib.dir/CXX.includecache
CFugue/test/CMakeFiles/testCFugueLib.dir/DependInfo.cmake
CFugue/test/CMakeFiles/testCFugueLib.dir/build.make
CFugue/test/CMakeFiles/testCFugueLib.dir/depend.internal
CFugue/test/CMakeFiles/testCFugueLib.dir/flags.make
CFugue/test/CMakeFiles/testCFugueLib.dir/link.txt
CFugue/test/Makefile
CFugue/test/cmake_install.cmake
```

8.1.2 Makefile

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

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : sb.native

sb.native :
        ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
                sb.native

# "make clean" removes all generated files

.PHONY : clean
clean :
        ocamlbuild -clean
        rm -rf testall.log *.diff sb scanner.ml parser.ml parser.mli
        rm -rf *.cmx *.cmi *.cmo *.cmx *.o
        rm *.ll *.out *.err *.dump *.dumpc *.midi

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx sb.cmx

sb : $(OBJS)
        ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis
          $(OBJS) -o sb

scanner.ml : scanner.mll
        ocamllex scanner.mll

parser.ml parser.mli : parser.mly
        ocamlyacc parser.mly

%.cmo : %.ml
        ocamlc -c $<

%.cmi : %.mli
        ocamlc -c $<

%.cmx : %.ml
        ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
```

```
45 ast.cmo :
46 ast.cmx :
47 codegen.cmo : ast.cmo
48 codegen.cmx : ast.cmx
49 sb.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
50 sb.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
51 parser.cmo : ast.cmo parser.cmi
52 parser.cmx : ast.cmx parser.cmi
53 scanner.cmo : parser.cmi
54 scanner.cmx : parser.cmx
55 semant.cmo : ast.cmo
56 semant.cmx : ast.cmx
57 parser.cmi : ast.cmo
58 preprocessor.cmo : ast.cmo
59 preprocessor.cmx : ast.cmx
60
61 # Building the tarball
62
63 TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3       \
64     func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3      \
65     hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2         \
66     while1 while2
67
68 FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2       \
69     for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8      \
70     func9 global1 global2 if1 if2 if3 nomain return1 return2 while1     \
71     while2
72
73 TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
74             $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)
75
76 TARFILES = ast.ml codegen.ml Makefile sb.ml parser.mly README scanner.mll \
77         semant.ml testall.sh $(TESTFILES:%=tests/%)
78
79 sb-llvm.tar.gz : $(TARFILES)
80         cd .. && tar czf sb-llvm/sb-llvm.tar.gz \
81                 $(TARFILES:%=sb-llvm/%)
```

8.1.3 README.md

```
# Sick Beets

### Sick Beets is a programming language that allows users to compose music by
generating MIDI files. Sick Beets is inspired by the structured nature of
music, which makes it easy to represent a composition piece by defining
```

```
attributes of notes such as pitch or duration. Using Sick Beets, users can
concatenate and layer series of notes to digitally encode their compositions.

## What is Sick Beets?
The language revolves around the idea of tunes, which are a series of notes
attached to durations. Users can easily put together arrays of notes and
durations to create a tune. In addition, users can concatenate tunes together
as well as layer them on top of each other. Through a series of built in
functions and operators, users are able to set the tempo, use different
instruments, and layer their tunes together. Finally, users can render their
tunes into MIDI files, which can be played to hear the final product.

## Running the Compiler and Executing a Program

1. Compile the CFugue Library (Make sure you delete the CFugueCache.txt file)
```shell
cd CFugue
cmake CMakeLists.txt
make
cd ..
```

2. Make the compiler
```shell
make
```

3. Compile the .sb file into llvm and run the llvm code
```shell
./sb.native < test.sb > test.ll
lli test.ll
```
```

8.1.4 scanner.ml

```
 1 { open Parser }
 2
 3 let un_letter = ['a' - 'z']
 4 let cap_letter = ['A' - 'Z']
 5 let letter = ['a' - 'z' 'A' - 'Z']
 6 let digit = ['0' - '9']
 7 let duration = 'w'|'h'|'q'|'i'|'s'|'t'
 8
 9 rule token = parse
10   [' ' '\t' '\r'] { token lexbuf }
11 | "/*"         { comment lexbuf }
12 | '('          { LPAREN }
```

```
 13 | ')'         { RPAREN }
 14 | '{'         { LBRACE }
 15 | '}'         { RBRACE }
 16 | '\n'        { EOL }
 17 | '+'         { PLUS }
 18 | '-'         { MINUS }
 19 | '*'         { TIMES }
 20 | '/'         { DIVIDE }
 21 | '%'         { MODULUS }
 22 | '<'         { LESS }
 23 | '>'         { GREATER }
 24 | '='         { ASSIGN }
 25 | ':'                 { COLON }
 26 | ';'               { SEMI }
 27 | '.'                 { DOT }
 28 | "<="        { LEQ }
 29 | ">="        { GEQ }
 30 | "=="        { EQ }
 31 | "!="        { NEQ }
 32 | "&&"        { AND }
 33 | "||"        { OR }
 34 | "!"                 { NOT }
 35 | "["         { LBRACKET }
 36 | "]"         { RBRACKET }
 37 | ","         { COMMA }
 38 | "if"        { IF }
 39 | "else"            { ELSE }
 40 | "while"     { WHILE }
 41 | "for"             { FOR }
 42 | "return"    { RETURN }
 43 | "bool"      { BOOL }
 44 | "float"     { FLOAT }
 45 | "string"    { STRING }
 46 | "int"       { INT }
 47 | "void"            { VOID }
 48 | "note"            { NOTE }
 49 | "duration"  { DURATION }
 50 | "tune"      { TUNE }
 51 | "char"      { CHAR }
 52 | "true"      { TRUE }
 53 | "false"     { FALSE }
 54 | eof         { EOF }
 55 | (['a'-'g'] ('b'|'#')? (['1'-'9'])?) | 'r' as note { NOTELIT note }
 56 | duration+ as dur { DURATIONLIT dur }
 57 | digit* '.' digit* (('e'|'E')('+'|'-')?digit+)? as flt {
FLOATLIT(float_of_string flt)}
 58 | "\"" ([^ '\"' ]* as str) "\"" { STRINGLIT(Scanf.unescaped str) }
 59 | digit+ as integer { INTLIT(int_of_string integer) }
```

```
60 | letter(letter | digit | '_')* as lxm { ID(lxm) }
61
62 and comment = parse
63  "*/" { token lexbuf }
64  | _ { comment lexbuf }
```

8.1.5 parser.mly

```
 1 %{ open Ast %}
 2
 3 %token LPAREN RPAREN LBRACE RBRACE
 4 %token PLUS MINUS TIMES DIVIDE ASSIGN MODULUS
 5 %token LESS GREATER LEQ GEQ EQ NEQ AND OR EOF NOT NEG COLON SEMI DOT
 6 %token RETURN WHILE IF ELIF ELSE FOR
 7 %token EOL
 8 %token BOOL INT STRING FLOAT VOID NOTE DURATION TUNE CHAR
 9 %token <int> INTLIT
10 %token <string> STRINGLIT
11 %token <float> FLOATLIT
12 %token <string> ID
13 %token <string> NOTELIT
14 %token <string> DURATIONLIT
15 %token <string> TUNELIT
16 %token TRUE FALSE
17 %token LBRACKET RBRACKET COMMA
18
19 %nonassoc NOELSE
20 %nonassoc ELSE
21
22 %right ASSIGN REASSIGN
23 %nonassoc LBRACKET RBRACKET
24 %left OR
25 %left AND
26 %left LESS GREATER LEQ GEQ
27 %left EQ NEQ
28 %left PLUS MINUS
29 %left TIMES DIVIDE MODULUS
30 %left DOT
31 %left AT
32 %left COLON
33 %left DOLLAR
34 %right NOT NEG
35 %nonassoc RETURN
36 %nonassoc IF
37
38 %start program
39 %type <Ast.program> program
40
```

```
41
42 %%
43
44
45 program: decls EOF { $1 }
46
47 decls:
48   /* nothing */   { [], [] }
49 | decls stmt      { (fst $1)@[$2], snd $1 }
50 | decls func_decl { fst $1, ($2 :: snd $1) }
51
52 stmts:
53   /* nothing  */  { [] }
54 | stmts stmt      { $2 :: $1 }
55
56 stmt:
57 | WHILE LPAREN expr RPAREN stmt         { While($3, $5) }
58 | LBRACE stmts RBRACE EOL               { Block(List.rev $2) }
59 | RETURN expr EOL                       { Return $2 }
60 | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
61 | IF LPAREN expr RPAREN stmt ELSE stmt  { If($3, $5, $7) }
62 | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt { For($3, $5, $7, $9) }
63 | expr EOL                              { Expr($1) }
64 | EOL                                   { Nostmt }
65
66
67 expr:
68   INTLIT        { IntLit($1) }
69 | FLOATLIT      { FloatLit($1) }
70 | STRINGLIT     { StringLit($1) }
71 | TRUE          { BoolLit(true) }
72 | FALSE         { BoolLit(false) }
73 | ID            { Id($1) }
74 | NOTELIT       { NoteLit($1) }
75 | DURATIONLIT   { DurationLit($1) }
76 | TUNELIT       { TuneLit($1) }
77 | expr PLUS expr    { Binop($1, Add, $3) }
78 | expr DOT expr     { Concat($1, $3) }
79 | expr MINUS expr   { Binop($1, Sub, $3) }
80 | expr COLON expr   { Augment($1, $3) }
81 | expr TIMES expr   { Binop($1, Mul, $3) }
82 | expr DIVIDE expr  { Binop($1, Div, $3) }
83 | expr MODULUS expr { Binop($1, Mod, $3) }
84 | expr LESS expr    { Binop($1, Lthan, $3) }
85 | expr GREATER expr { Binop($1, Gthan, $3) }
86 | expr LEQ expr     { Binop($1, Leq, $3) }
87 | expr GEQ expr     { Binop($1, Geq, $3) }
88 | expr EQ expr      { Binop($1, Eq, $3) }
```

```
 89 | expr NEQ expr     { Binop($1, Neq, $3) }
 90 | expr AND expr     { Binop($1, And, $3) }
 91 | expr OR expr      { Binop($1, Or, $3) }
 92 | MINUS expr %prec NEG  { Unop(Neg, $2) }
 93 | NOT expr              { Unop(Not, $2) }
 94 | typ ID ASSIGN expr    { Assign($1, $2, $4) }
 95 | ID ASSIGN expr        { Reassign($1, $3) }
 96 | ID LPAREN actuals_opt RPAREN          { Call($1, $3) }
 97 | LBRACKET elements_opt RBRACKET        { ArrayLit($2) }
 98 | primitive LBRACKET expr RBRACKET      { ArrayLitDefault($1, $3) }
 99 | ID LBRACKET expr RBRACKET             { ArrayAccess($1, $3) }
100 | ID LBRACKET expr RBRACKET ASSIGN expr { ArrayElementAssign($1, $3, $6) }
101
102 typ:
103   primitive                   { $1 }
104 | primitive LBRACKET RBRACKET { Array($1) }
105
106 primitive:
107   INT       { Int }
108 | FLOAT     { Float }
109 | BOOL      { Bool }
110 | STRING    { String }
111 | VOID      { Void }
112 | NOTE      { Note }
113 | DURATION  { Duration }
114 | TUNE      { Tune }
115 | CHAR      { Char}
116
117 actuals_opt:
118   /* nothing*/  { [] }
119 | actuals_list  { List.rev $1 }
120
121 actuals_list:
122   expr                    { [$1] }
123 | actuals_list COMMA expr { $3 :: $1 }
124
125 func_decl:
126   typ ID LPAREN formals_opt RPAREN LBRACE stmts RBRACE
127   {{  typ = $1;
128       fname = $2;
129       formals = $4;
130       body = List.rev $7 }}
131
132 formals_opt:
133    /*nothing*/  { [] }
134 | formal_list   { List.rev $1 }
135
136 formal_list:
```

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

elements_opt:
  /*nothing*/   { [] }
| elements_list { List.rev $1 }

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

```

8.1.6 ast.ml

```
type operator = Add | Sub | Mul | Div | Mod
              | Leq | Geq | Lthan | Gthan
              | Eq | Neq | And | Or

type uop = Neg | Not

type typ = Bool | Int | Char | String | Float | Void | Note | Duration | Tune
             | Array of typ

type bind = typ * string

type expr =
    IntLit of int
  | BoolLit of bool
  | StringLit of string
  | FloatLit of float
  | Binop of expr * operator * expr
  | Unop of uop * expr
  | Id of string
  | NoteLit of string
  | DurationLit of string
  | TuneLit of string
  | Call of string * expr list
  | Assign of typ * string * expr
  | Reassign of string * expr
  | ArrayLit of expr list
  | ArrayLitDefault of typ * expr
  | ArrayAccess of string * expr
  | ArrayElementAssign of string * expr * expr
  | Augment of expr * expr
  | Concat of expr * expr
  | Noexpr

```

```
33 type stmt =
34     Block of stmt list
35   | While of expr * stmt
36   | Return of expr
37   | If of expr * stmt * stmt
38   | For of expr * expr * expr * stmt
39   | Expr of expr
40   | Nostmt
41
42 type func_decl = {
43         typ : typ;
44         fname : string;
45         formals : bind list;
46         body : stmt list;
47 }
48
49 type program = stmt list * func_decl list
50
51 let rec string_of_typ = function
52     Int -> "int"
53   | Bool -> "bool"
54   | String -> "string"
55   | Float -> "float"
56   | Void -> "void"
57   | Note -> "note"
58   | Duration -> "duration"
59   | Array t -> string_of_typ t ^ "[]"
60   | Tune -> "tune"
61   | Char -> "char"
62
63 let string_of_op = function
64     Add -> "+"
65   | Sub -> "-"
66   | Mul -> "*"
67   | Div -> "/"
68   | Eq -> "=="
69   | Neq -> "!="
70   | Lthan -> "<"
71   | Leq -> "<="
72   | Gthan -> ">"
73   | Geq -> ">="
74   | And -> "&&"
75   | Or -> "||"
76   | Mod -> "%"
77
78 let string_of_uop = function
79     Neg -> "-"
80   | Not -> "!"
```

```
 81
 82 let rec string_of_expr = function
 83     StringLit(l) ->  l
 84   | IntLit(i) -> string_of_int i
 85   | BoolLit(true) -> "true"
 86   | BoolLit(false) -> "false"
 87   | FloatLit(f) -> string_of_float f
 88   | NoteLit(n) -> n
 89   | DurationLit(d) -> d
 90   | TuneLit(t) -> t
 91   | Id(s) -> s
 92   | Unop(o, e) -> string_of_uop o ^ string_of_expr e
 93   | Assign(t, v, e) -> string_of_typ t ^ " " ^ v ^ " = " ^ string_of_expr e
 94   | Call(f, el) ->
 95     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
 96   | Binop(e1, o, e2) ->
 97     string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
 98   | Reassign(v, e) -> v ^ "=" ^ string_of_expr e
 99   | ArrayLit a -> "[" ^ String.concat " " (List.map string_of_expr a) ^ "]"
100   | ArrayLitDefault(t, e) -> string_of_typ t ^ "[" ^ string_of_expr e ^ "]"
101   | Noexpr -> ""
102   | ArrayAccess(v, i) -> v ^ "[" ^ string_of_expr i ^ "]"
103   | ArrayElementAssign(s, i, e) -> s ^ "[" ^ string_of_expr i ^ "]" ^ " = " ^
string_of_expr e
104   | Augment(l, r) -> string_of_expr l ^ ":" ^ string_of_expr r
105   | Concat(l, r) -> string_of_expr l ^ "." ^ string_of_expr r
106
107 let rec string_of_stmt = function
108     Block(stmts) ->
109       "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
110   | Expr(expr) -> string_of_expr expr ^ "\n";
111   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt
s
112   | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
113       string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
114   | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
115   | Return(expr) -> "return " ^ string_of_expr expr ^ "\n"
116   | For(e1, e2, e3, s) ->
117       "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
118       string_of_expr e3  ^ ") " ^ string_of_stmt s
119   | Nostmt -> ""
120
121 let string_of_fdecl fdecl =
122   string_of_typ fdecl.typ ^ " " ^
123   fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
124   ")\n{\n" ^
125   String.concat "" (List.map string_of_stmt fdecl.body) ^
126   "}\n"
```

```
127
128 let string_of_program (funcs) =
129   String.concat "\n" (List.map string_of_fdecl funcs)
130
```

8.1.7 preprocessor.ml

1.

```
 1 (* Preprocessor for the sick-beets compiler *)
 2
 3 open Ast
 4
 5 module StringMap = Map.Make(String)
 6
 7 (* Semantic checking of a program. Returns void if successful,
 8    throws an exception if something is wrong.
 9 :x
10    Check each global variable, then check each function *)
11
12
13
14
15 let process (statements, functions) =
16
17   let main_function =
18     {
19       typ = Ast.Int;
20       fname = "main";
21       formals = [];
22       body = statements;
23     }
24   in main_function :: functions
25
```

8.1.8 semant.ml

8.1.9 codegen.ml

```
 1 module L = Llvm
 2 module A = Ast
 3 module S = Semant
 4
 5 module StringMap = Map.Make(String)
 6
 7 let translate (functions) =
 8
```

```
  9   (* Set LLVM Types *)
 10   let context = L.global_context () in
 11   let the_module = L.create_module context "sick-beets"
 12   and i32_t  = L.i32_type  context
 13   and i8_t   = L.i8_type   context
 14   and i1_t   = L.i1_type   context
 15   and f_t = L.double_type context
 16   and str_t = L.pointer_type (L.i8_type context)
 17   and note_t = L.pointer_type (L.i8_type context)
 18   and dur_t = L.pointer_type (L.i8_type context)
 19   and tune_t = L.pointer_type (L.i8_type context)
 20   and void_t = L.void_type context in
 21
 22   (* Mapping for AST type to LLVM types *)
 23   let rec ltype_of_typ = function
 24       A.Int -> i32_t
 25     | A.Float -> f_t
 26     | A.String -> str_t
 27     | A.Note -> note_t
 28     | A.Tune -> tune_t
 29     | A.Duration -> dur_t
 30     | A.Char -> i8_t
 31     | A.Bool -> i1_t
 32     | A.Void -> void_t
 33     | A.Array t -> L.pointer_type (ltype_of_typ t) in
 34
 35   (* Array creation, initialization, access *)
 36   let create_array t len builder =
 37     let ltype = ltype_of_typ t in
 38     let size_t = L.build_intcast (L.size_of ltype) i32_t "tmp" builder in
 39     let total_size = L.build_mul size_t len "tmp" builder in
 40     let total_size = L.build_add total_size (L.const_int i32_t 1) "tmp"
builder in
 41     let arr_malloc = L.build_array_malloc ltype total_size "tmp" builder in
 42     let arr = L.build_pointercast arr_malloc (L.pointer_type ltype) "tmp"
builder in
 43     arr
 44   in
 45
 46   let initialize_array t el builder =
 47     let len = L.const_int i32_t (List.length el) in
 48     let arr = create_array t len builder in
 49     let _ =
 50       let assign_value i =
 51         let index = L.const_int i32_t i in
 52         let index = L.build_add index (L.const_int i32_t 1) "tmp" builder in
 53         let _val = L.build_gep arr [| index |] "tmp" builder in
 54         L.build_store (List.nth el i) _val builder
```

```
 55       in
 56       for i = 0 to (List.length el)-1 do
 57         ignore (assign_value i)
 58       done
 59     in
 60     arr
 61   in
 62
 63   let access_array arr index assign builder =
 64     let index = L.build_add index (L.const_int i32_t 1) "tmp" builder in
 65     let arr = L.build_load arr "tmp" builder in
 66     let _val = L.build_gep arr [| index |] "tmp" builder in
 67     if assign then _val else L.build_load _val "tmp" builder
 68   in
 69
 70   (* initialize global variable map from Ast.Id to llvalue *)
 71   let global_vars = ref StringMap.empty in
 72
 73   (* Declare and define printf, strcat, strcpy *)
 74   let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
 75   let printf_func = L.declare_function "printf" printf_t the_module in
 76
 77   let strcat_t = L.function_type (L.pointer_type i8_t) [| L.pointer_type i8_t;
L.pointer_type i8_t |] in
 78   let strcat_func = L.declare_function "strcat" strcat_t the_module in
 79
 80   let strcpy_t = L.function_type (L.pointer_type i8_t) [| L.pointer_type i8_t;
L.pointer_type i8_t |] in
 81   let strcpy_func = L.declare_function "strcpy" strcpy_t the_module in
 82
 83   (* create a map of function declarations *)
 84   let function_decls =
 85     let function_decl m fdecl =
 86       let name = fdecl.A.fname
 87       and formal_types =
 88         Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
in
 89       let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types in
 90       StringMap.add name (L.define_function name ftype the_module, fdecl) m in
 91       List.fold_left function_decl StringMap.empty functions in
 92
 93   (* Format str for printf *)
 94   let string_format_str b = L.build_global_stringptr "%s" "fmt" b
 95   and int_format_str    b = L.build_global_stringptr "%d" "fmt" b
 96   and float_format_str  b = L.build_global_stringptr "%f" "fmt" b in
 97
 98   let format_str x_type builder =
 99     let b = builder in
```

```
100       match x_type with
101         A.Int      -> int_format_str b
102       | A.Float    -> float_format_str b
103       | A.String | A.Note | A.Duration | A.Tune  -> string_format_str b
104       | A.Bool     -> int_format_str b
105       | _ -> raise (Failure ("Invalid printf type"))
106   in
107
108   (* build the statements in a function body *)
109   let build_function_body fdecl =
110     let (f, _) = StringMap.find fdecl.A.fname function_decls in
111     let builder = L.builder_at_end context (L.entry_block f) in
112
113     (* function to add local variables to a map *)
114     let local_vars =
115       let add_formal m (t,n) p = L.set_value_name n p;
116       let local = L.build_alloca (ltype_of_typ t) n builder in
117       ignore (L.build_store p local builder);
118       StringMap.add n (t,local) m in
119
120       (* add formals to the map *)
121       ref (List.fold_left2 add_formal StringMap.empty fdecl.A.formals
122         (Array.to_list (L.params f))) in
123
124     (* Return the value or the type for a variable or formal argument *)
125     (* All the tables have the structure (type, llvalue) *)
126     let name_to_llval n : L.llvalue =
127       try (snd (StringMap.find n !local_vars))
128       with Not_found -> (snd (StringMap.find n !global_vars))
129     in
130
131     let name_to_type n : A.typ =
132       try (fst (StringMap.find n !local_vars))
133       with Not_found -> (fst (StringMap.find n !global_vars)) in
134
135     (* get type *)
136     let rec gen_type = function
137         A.IntLit _ -> A.Int
138       | A.FloatLit _ -> A.Float
139       | A.StringLit _ -> A.String
140       | A.BoolLit _ -> A.Bool
141       | A.ArrayLit el -> A.Array (gen_type (List.nth el 0))
142       | A.ArrayLitDefault (t,_) -> t
143       | A.ArrayElementAssign (_, _, el) -> gen_type el
144       | A.NoteLit _ -> A.Note
145       | A.DurationLit _ -> A.Duration
146       | A.TuneLit _ -> A.Tune
147       | A.Id name -> (match (name_to_type name) with
```

```
148                       A.Array(t) -> t
149                     | _ as ty -> ty)
150       | A.Unop(_,e) -> gen_type e
151       | A.Binop(e1,_,_) -> gen_type e1
152       | A.Assign(_,var,_) -> gen_type (A.Id(var))
153       | A.Reassign(var,_) -> gen_type (A.Id(var))
154       | A.Call(var,_) -> let fdecl =
155                             List.find (fun x -> x.A.fname = var) functions in
156                             fdecl.A.typ
157       | A.ArrayAccess(id,_) -> gen_type (A.Id(id))
158       | A.Augment(_,_) |  A.Concat(_,_) -> A.Tune
159       | A.Noexpr -> A.Void
160     in
161
162     (* get value of int *)
163     let get_value_of_int = function
164         A.IntLit x -> x
165       | _ -> raise (Failure ("can only get_value_of_int of an int"))
166     in
167
168     (* Declare system functions we need *)
169     let system_t = L.function_type i32_t [| str_t |] in
170     let system_func = L.declare_function "system" system_t the_module in
171
172     (* build string *)
173     let build_string s n builder =
174     let zero_const = L.const_int i32_t 0 in
175       let str = L.build_in_bounds_gep s [| zero_const |] " " builder in
176       L.build_call system_func [| str |] n builder
177     in
178
179     (* expression builder *)
180     let rec expr builder = function
181         A.IntLit i -> L.const_int i32_t i
182       | A.StringLit s -> L.build_global_stringptr s "string" builder
183       | A.FloatLit f -> L.const_float f_t f
184       | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
185       | A.ArrayLit el -> let t = gen_type (List.nth el 0) in
186         initialize_array t (List.map (expr builder) el) builder
187       | A.ArrayLitDefault (t, e) -> let len = expr builder e in
188         create_array t len builder
189       | A.NoteLit n -> L.build_global_stringptr n "note" builder
190       | A.DurationLit d -> L.build_global_stringptr d "duration" builder
191       | A.TuneLit t -> L.build_global_stringptr t "tune" builder
192       | A.Id s -> let llval = (name_to_llval s) in
193                   L.build_load llval s builder
194       | A.Noexpr -> L.const_int i32_t 0
195       | A.Unop(op,e) -> let e' = expr builder e in
```

```
196         (match op with
197           A.Neg     -> L.build_neg
198           | A.Not     -> L.build_not) e' "tmp" builder
199       | A.Binop (e1, op, e2) ->
200         let e1' = expr builder e1
201         and e2' = expr builder e2
202         and float_binops =
203           match op with
204             A.Add -> L.build_fadd
205           | A.Sub -> L.build_fsub
206           | A.Mul -> L.build_fmul
207           | A.Div -> L.build_fdiv
208           | A.Mod     -> L.build_frem
209           | A.And     -> L.build_and
210           | A.Or      -> L.build_or
211           | A.Eq -> L.build_fcmp L.Fcmp.Oeq
212           | A.Neq -> L.build_fcmp L.Fcmp.One
213           | A.Lthan -> L.build_fcmp L.Fcmp.Ult
214           | A.Leq -> L.build_fcmp L.Fcmp.Ole
215           | A.Gthan -> L.build_fcmp L.Fcmp.Ogt
216           | A.Geq -> L.build_fcmp L.Fcmp.Oge
217
218         and int_binops =
219           match op with
220             A.Add -> L.build_add
221           | A.Sub -> L.build_sub
222           | A.Mul -> L.build_mul
223           | A.Div -> L.build_sdiv
224           | A.Mod     -> L.build_urem
225           | A.And     -> L.build_and
226           | A.Or      -> L.build_or
227           | A.Eq -> L.build_icmp L.Icmp.Eq
228           | A.Neq -> L.build_icmp L.Icmp.Ne
229           | A.Lthan -> L.build_icmp L.Icmp.Slt
230           | A.Leq -> L.build_icmp L.Icmp.Sle
231           | A.Gthan -> L.build_icmp L.Icmp.Sgt
232           | A.Geq -> L.build_icmp L.Icmp.Sge
233         in
234
235          if (L.type_of e1' = f_t || L.type_of e2' = f_t) then float_binops e1'
e2' "tmp" builder
236          else int_binops e1' e2' "tmp" builder
237
238       | A.Call ("print", [e]) ->
239           let e' = expr builder e in
240           let e_type = gen_type e in
241           L.build_call printf_func [| (format_str e_type builder) ; e' |]
"printf" builder
```

```
242       | A.Call ("render", [e1; e2]) ->
243           let exec = expr builder (A.StringLit("CFugue/bin/testCFugueLib \""))
244           and e1' = expr builder e1
245           and quote = expr builder (A.StringLit("\" "))
246           and e2' = expr builder e2 in
247           let buf = expr builder (A.ArrayLitDefault(A.Char, A.IntLit(4096)))
in
248           let tmp1 = L.build_call strcpy_func [| buf; exec |] "strcpy" builder
in
249           let tmp2 = L.build_call strcat_func [| tmp1; e1' |] "strcat" builder
in
250           let tmp3 = L.build_call strcat_func [| tmp2; quote |] "strcat"
builder in
251           let render_cmd = L.build_call strcat_func [| tmp3; e2' |] "strcat"
builder in
252           build_string render_cmd "rendf" builder;
253       | A.Call ("setInstrument", [e1; e2]) ->
254           let tune = expr builder e1 and instr = expr builder e2 in
255           let left = expr builder (A.StringLit("I[")) in
256           let right = expr builder (A.StringLit("] ")) in
257           let buf = expr builder (A.ArrayLitDefault(A.Char, A.IntLit(4096)))
in
258           let tmp1 = L.build_call strcpy_func [| buf; left |] "tmp1" builder
in
259           let tmp2 = L.build_call strcat_func [| tmp1; instr |] "tmp2" builder
in
260           let tmp3 = L.build_call strcat_func [| tmp2; right |] "tmp3" builder
in
261         L.build_call strcat_func [| tmp3; tune |] "strcat" builder
262       | A.Call ("addLayer", [e1; e2; e3]) ->
263           let ltune = expr builder e1 and rtune = expr builder e2 and layer =
expr builder e3 in
264           let channel = expr builder (A.StringLit("V")) in
265           let space = expr builder (A.StringLit(" ")) in
266           let buf = expr builder (A.ArrayLitDefault(A.Char, A.IntLit(4096)))
in
267           let tmp1 = L.build_call strcpy_func [| buf; ltune |] "tmp1" builder
in
268           let tmp2 = L.build_call strcat_func [| tmp1; channel |] "tmp2"
builder in
269           let tmp3 = L.build_call strcat_func [| tmp2; layer |] "tmp3" builder
in
270           let tmp4 = L.build_call strcat_func [| tmp3; space |] "tmp4" builder
in
271           L.build_call strcat_func [| tmp4; rtune |] "strcat" builder
272       | A.Call ("setTempo", [e1; e2]) ->
273           let tune' = expr builder e1
```

```
274           and e' = expr builder (A.StringLit((string_of_int (get_value_of_int
e2))))
275           and space' = expr builder (A.StringLit(" "))
276           and t' = expr builder (A.StringLit("T")) in
277           let buf = expr builder (A.ArrayLitDefault(A.Char, A.IntLit(4096)))
in
278           let _ = L.build_call strcpy_func [| buf; t' |] "partial" builder in
279           let _ = L.build_call strcat_func [| buf; e' |] "partial_2" builder
in
280           let _ = L.build_call strcat_func [| buf; space' |] "partial_3"
builder in
281           L.build_call strcat_func [| buf; tune' |] "strcat" builder
282       | A.Call (f, act) ->
283           let (fdef, fdecl) = StringMap.find f function_decls in
284           let actuals = List.rev (List.map (expr builder) (List.rev act)) in
285           let result = (match fdecl.A.typ with
286              A.Void -> ""
287             | _ -> f ^ "_result") in
288           L.build_call fdef (Array.of_list actuals) result builder
289       | A.Assign (t, s, e) ->
290           let _ = (match fdecl.A.fname with
291             "main" ->
292               let init = (match t with
293                 A.Int   -> expr builder (A.IntLit(0))
294               | A.Float -> expr builder (A.FloatLit(0.0))
295               | A.String -> expr builder (A.StringLit(""))
296               | A.Bool   -> expr builder (A.BoolLit(false))
297               | A.Note -> expr builder (A.NoteLit(""))
298               | A.Duration -> expr builder (A.DurationLit(""))
299               | A.Tune -> expr builder (A.TuneLit(""))
300               | t -> L.const_null (ltype_of_typ t)
301             )
302           in
303                 (global_vars := StringMap.add s (t, (L.define_global s init
the_module)) !global_vars)
304             | _ -> (local_vars := StringMap.add s (t, (L.build_alloca
(ltype_of_typ t)) s builder) !local_vars)
305             ) in
306           let e' = expr builder e and llval = name_to_llval s in
307           ignore (L.build_store e' llval builder); e'
308       | A.Reassign (s, e) ->
309           let e' = expr builder e and llval = name_to_llval s in
310           ignore (L.build_store e' llval builder); e'
311       | A.ArrayAccess (s, i) ->
312           let index = expr builder i and llval = name_to_llval s in
313           access_array llval index false builder
314       | A.ArrayElementAssign (s, i, e) ->
315           let e' = expr builder e in
```

```
316           let index = expr builder i in
317           let llval = name_to_llval s in
318           let var = access_array llval index true builder in
319           ignore (L.build_store e' var builder); e'
320       | A.Augment(e1, e2) ->
321           let space = expr builder (A.StringLit(" ")) in
322           let empty = expr builder (A.StringLit("")) in
323
324           (match (e1,e2) with
325             (A.NoteLit _, A.DurationLit _) ->
326               let note = expr builder e1 and dur = expr builder e2 in
327               let buf = expr builder (A.ArrayLitDefault(A.Char,
A.IntLit(4096))) in
328               let space = expr builder (A.StringLit(" ")) in
329               let _ = L.build_call strcpy_func [| buf; note |] "tmp1" builder
in
330               let _ = L.build_call strcat_func [| buf; dur |] "tmp2" builder
in
331               L.build_call strcat_func [| buf; space|] "strcat" builder
332
333             | (A.NoteLit _, A.ArrayLit a) ->
334               let x = expr builder e1 in
335               let buf = expr builder (A.ArrayLitDefault(A.Char,
A.IntLit(4096))) in
336               let buf = L.build_call strcpy_func [| buf; empty |] "tmp"
builder in
337               let _ = List.fold_left (fun s e ->
338                 let e' = expr builder e in
339                 let _ = L.build_call strcat_func[| s; x|] "tmp2" builder in
340                 let _ = L.build_call strcat_func[| s; e'|] "tmp3" builder in
341                 L.build_call strcat_func [| s; space |] "strcat" builder ) buf
a  in
342               L.build_call strcat_func [| buf; space |] "strcat" builder
343
344             | (A.ArrayLit a, A.DurationLit _) ->
345               let x = expr builder e2 in
346               let buf = expr builder (A.ArrayLitDefault(A.Char,
A.IntLit(4096))) in
347               let buf = L.build_call strcpy_func [| buf; empty |] "tmp"
builder in
348               let _ = List.fold_left (fun s e ->
349                 let e' = expr builder e in
350                 let _ = L.build_call strcat_func[| s; e'|] "tmp2" builder in
351                 let _ = L.build_call strcat_func[| s; x|] "tmp3" builder in
352                 L.build_call strcat_func [| s; space |] "strcat" builder ) buf
a  in
353               L.build_call strcat_func [| buf; space |] "strcat" builder
354
```

```
355             | (A.ArrayLit n, A.ArrayLit d) ->
356               let buf = expr builder (A.ArrayLitDefault(A.Char,
A.IntLit(4096))) in
357               let buf = L.build_call strcpy_func [| buf; empty |] "tmp"
builder in
358               let _ = List.fold_left2 (fun s e1 e2 ->
359                 let e1' = expr builder e1 in
360                 let e2' = expr builder e2 in
361                 let _ = L.build_call strcat_func[| s; e1'|] "tmp2" builder in
362                 let _ = L.build_call strcat_func[| s; e2'|] "tmp3" builder in
363                 L.build_call strcat_func [| s; space |] "strcat" builder ) buf
n d  in
364               L.build_call strcat_func [| buf; space |] "strcat" builder
365
366             | (A.Id _, A.Id _) | (A.NoteLit _, A.Id _) | (A.Id _,
A.DurationLit _) ->
367               let e1' = expr builder e1 and e2' = expr builder e2 in
368               let buf = expr builder (A.ArrayLitDefault(A.Char,
A.IntLit(4096))) in
369               let tmp = L.build_call strcpy_func [| buf; e1'|] "tmp" builder
in
370
371               L.build_call strcat_func [| tmp; e2'|] "strcat" builder
372             | (_, _) -> expr builder (A.TuneLit(""))
373           )
374
375       | A.Concat(e1, e2) ->
376           let e1' = expr builder e1 and e2' = expr builder e2 in
377           let space = expr builder (A.StringLit(" ")) in
378           let buf = expr builder (A.ArrayLitDefault(A.Char, A.IntLit(4096)))
in
379           let tmp1 = L.build_call strcpy_func [| buf; e1' |] "tmp1" builder in
380           let tmp2 = L.build_call strcat_func [| tmp1; e2' |] "tmp2" builder
in
381           L.build_call strcat_func [|tmp2;space |] "strcat" builder
382       in
383
384     (* Invoke "f builder" if the current block doesn't already
385        have a terminal (e.g., a branch). *)
386     let add_terminal builder' f' =
387       match L.block_terminator (L.insertion_block builder') with
388           Some _ -> ()
389         | None -> ignore(f' builder') in
390
391     (* Build the code for the given statement; return the builder for
392        the statement's successor *)
393     let rec stmt builder = function
394         A.Block sl -> List.fold_left stmt builder sl
```

```
395       | A.Expr e -> ignore(expr builder e); builder
396       | A.Nostmt -> ignore (0); builder
397       | A.Return e -> ignore (match fdecl.A.typ with
398          A.Void -> L.build_ret_void builder
399          | _ -> L.build_ret (expr builder e) builder); builder
400       | A.If (predicate, then_stmt, else_stmt) ->
401         let bool_val = expr builder predicate in
402         let merge_bb = L.append_block context "merge" f in
403
404         let then_bb = L.append_block context "then" f in
405         add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
406          (L.build_br merge_bb);
407
408         let else_bb = L.append_block context "else" f in
409         add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
410           (L.build_br merge_bb);
411
412         ignore (L.build_cond_br bool_val then_bb else_bb builder);
413         L.builder_at_end context merge_bb
414
415       | A.While (predicate, body) ->
416         let pred_bb = L.append_block context "while" f in
417         ignore (L.build_br pred_bb builder);
418
419         let body_bb = L.append_block context "while_body" f in
420         add_terminal (stmt (L.builder_at_end context body_bb) body)
421           (L.build_br pred_bb);
422
423         let pred_builder = L.builder_at_end context pred_bb in
424         let bool_val = expr pred_builder predicate in
425
426         let merge_bb = L.append_block context "merge" f in
427         ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
428         L.builder_at_end context merge_bb
429       | A.For (e1, e2, e3, body) -> stmt builder
430         ( A.Block [A.Expr e1; A.While (e2, A.Block [body; A.Expr e3]) ] )
431     in
432
433     (* Build the code for all statements in the function *)
434     let builder = stmt builder (A.Block fdecl.A.body) in
435
436     (* Add a return if the last block falls off the end *)
437     add_terminal builder (match fdecl.A.typ with
438         A.Int -> L.build_ret (L.const_int i32_t 0)
439       | A.Float -> L.build_ret (L.const_float f_t 0.0)
440       | A.Bool -> L.build_ret (L.const_int i1_t 0)
441       | A.String | A.Note | A.Duration | A.Tune -> L.build_ret (L.const_int
i8_t 0)
```

```
442       | _ -> L.build_ret_void) in
443
444   List.iter build_function_body functions ;
445
446   the_module
```

8.2.10 CFugue/test/StaticLibTestApp/SampleApp.cpp

```
 1 /*
 2         This is part of CFugue, a C++ Runtime for MIDI Score Programming
 3         Copyright (C) 2009 Gopalakrishna Palem
 4
 5         For links to further information, or to contact the author,
 6         see <http://cfugue.sourceforge.net/>.
 7
 8         This file has been edited by the SickBeets team in order to make an
               executable that we can utilize when rendering MIDI files.
 9 */
10
11 // SampleApp.cpp
12 //
13 // Demonstrates the usage of CFugue as a Statically Linked Library
14 //
15 // CFugue is under active development.
16 // Please refer to documentation and latest releases at:
http://cfugue.sourceforge.net/
17
18 #include "stdafx.h"
19 #include "stdlib.h"
20 #include <iostream>
21
22 #include "CFugueLib.h"
23
24 using namespace std;
25
26 void OnParseTrace(const CFugue::CParser*,
          CFugue::CParser::TraceEventHandlerArgs* pEvArgs)
27 {
28      wcout << "\n\t" << pEvArgs->szTraceMsg;
29 }
30
31 void OnParseError(const CFugue::CParser*,
          CFugue::CParser::ErrorEventHandlerArgs* pEvArgs)
32 {
33         wcerr << "\n\t Error --> " << pEvArgs->szErrMsg;
34         if(pEvArgs->szToken)
35         {
36                 wcerr << "\t Token: " << pEvArgs->szToken;
```

```
 37         }
 38 }
 39
 40 int main(int argc, char* argv[])
 41 {
 42
 43   if (argc != 3)
 44   {
 45     exit(fprintf(stderr, "CFugue App Arguments: [String: notes] [String: output
file name]"));
 46   }
 47
 48   CFugue::Player player;
 49         player.SaveAsMidiFile(argv[1], argv[2]);
 50
 51   string render_msg ("Rendered ");
 52   cout << render_msg + argv[1] << endl;
 53
 54   return 0;
 55
 56 }
 57
```