C♭

Spring 2021
Programming Languages and Translators

Yvonne Chen, Isabella Cho, Katherine Kim, Jasmine Valera
Professor Stephen A. Edwards

1. Introduction

C♭ is a computer language designed for musical composition. (The name of the language is pronounced [**SEE**-F*UH*LAT] and can also be written as “C-Flat” when special characters are not available.) Intended to be intuitive for users with a broad range of music theory knowledge, the language is fundamentally built upon and inspired by rudimentary elements of music theory. The language is statically scoped, strongly typed, statically typed, and complete with mutable and immutable, atomic and composite data types.

The following sections describe the many data types and methods especially unique to the musicality of C♭.

1.1. White Paper

The C♭ language is designed to enable ease in musical composition and audio synthesis directly from a computer keyboard — as though via a piano keyboard. Favoring both flexibility and readability, C♭ is a programming language that can be as musically powerful and technically efficient as the coder and composer themself. Within the realm of raised octaves, key changes, and semiquavers, there is no limit to creativity in C♭.

1.1.1. Musically Powerful

Being deceptively compact, features of the C♭ language are surprisingly minimal, but unbelievably powerful. The full suite of data types of the C♭ language, along with arrays, do provide for both a tangible coding experience and realistic control over composition.

1.1.2. Natural and Intuitive

C♭ is novel, with many original musical features. However, the chimeric C♭ music language also borrows syntactic aspects of C, and shares a functional resemblance with many object-oriented languages as well. Anyone familiar with coding in C, Java, or Python will experience a very natural transition to writing programs in C-Flat.

1.1.3. Music To Your Ears

Upon compilation, a C♭ program can generate a midi file. Midi files are the technical, standard file format for any playable or editable musical ambitions. Midi files are extremely versatile, and can be played using a DAW (Digital Audio Workstation, e.g. Logic Pro) or a Midi sequencer.

2. Language Tutorial

2.1. Environment Setup

This language requires the following prerequisites:

Dependency	Supported Version
Docker Engine	20.10.2
LLVM	10.0.0
OCaml	4.12.0
GCC	9.3

Once all prerequisites have been fulfilled, download all code, including all the tests in the `/tests` directory, from the repository on GitHub.

2.2. Compilation Guide

Enter the `/CFlat` directory to build the compiler. Invoke Docker using the following command:

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

Once inside Docker, first run `make clean` to make sure all pre-generated files are removed. Then run `make`, which will both build the compiler and run all tests. All tests should pass (indicated by `OK`).

Once you have a file with the format `.cf` containing C♭ code, run this command:

```
./cflat.native <filename>.cf
```

2.3. Programming Tutorial

Here is a walkthrough to create your first C♭ program! First, create a file with the format `.cf`, e.g. `helloworld.cf`. Copy the following code into it:

```
int main()
{
      note n;
      string file;
      n = ( /C-/ /4/ /s./ );
      printn(n);
```

```
    file = "output";
    playnote(n, file);
    return 0;
}
```

This program should print `/C-/ /4/ /s./` to the standard output, and also create a MIDI file named `output.mid` in your current `/CFlat` directory. Try playing the generated MIDI: you could use this online tool: https://onlinesequencer.net/. You should hear a 1/16 C-flat 4 note, with the default tempo of 120 bpm!

3. Language Manual

3.1. Lexical Conventions

In the following sections, "regex" refers to a regular expression. Specifically, that which was used to tokenize certain lexemes in the scanner (Ocamllex) of the C♭ compiler. (Note that in Ocamllex, if there exists more than one regex that matches the prefix of an input, the longest match takes precedence. Moreover, "-" between two single characters represent a range of characters. "^" indicates concatenation.)

3.1.1. Comments

Any sequence of characters that begin with the characters "`(:`" and end with the characters "`:)`" is considered a comment. Comments may extend over any number of lines in the code.

3.1.2. Identifiers

Identifiers are simply sequences of characters. As a rule, the first character of an identifier must be alphabetic or the "_" character in order to be a valid identifier. Identifiers are also case-sensitive. The purpose of an identifier in a C♭ program is to identify a function or some constant.

The following regex regulates identifiers: `['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`.

3.1.3. Keywords

All sequences of characters that are shown below are reserved with a specified significance in the C♭ language, and may never be used as identifiers.

```
if      for     bool    else     note     make    false      float      rhythm
int     def     true    tone     true     void    while      string     octave
```

`.tone`	`.octave`	`.rhythm`	`.raiseOctave()`
`.tone()`	`.octave()`	`.rhythm()`	`.lowerOctave()`

3.1.4. Literals

3.1.4.1. Integer Literals

Integer literals are sequences of numeric symbols treated as decimal digits. Integers in C♭ are 2 bytes. Both positive and negative integers are allowed from -32,768 to 32,767. Integer literals are defined by this regex: `['0' - '9']+`.

3.1.4.2. Floating Point Literals

Floating point literals consist of an integer part, a decimal character, and a fractional part. The integer part and fractional parts are observed as sequences of numeric symbols, like integers. No part of the floating point literal may be exempt from declaration. Floating point literals are always computed single-precision, meaning 4 bytes are used, and are defined by the following regular expression: `digits '.' digit* ( ['e' 'E'] ['+' '-']? digits )?` where `digit = ['0' - '9']` and `digits = digit+`.

3.1.4.3. Strings Literals

String literals are recognized in the C♭ language, enclosed in double quotation marks “"”. String literals may be of any length. In the scanner, they are determined as so: `'"' ((ascii | esc)* as s) '"'`.

In the aforementioned regex, `esc = '\\' ['\\' ''' '"' 'n' 'r' 't']` and similarly, `ascii = ([' '-'!' '#'-'[' ']'-'~'])`.

3.1.4.4. Boolean Literals

The two boolean literal constants as supported and represented in C♭ are “`false`” and “`true`” — and nothing else.

3.1.4.5. Tone Literals

The tone literals are unique to C♭. They are observed by the following regex: `'/' (((['A'-'G']['+''-'.']?)|'R') '/'`. In the following sections, the significance of a tone literal, and its uses, as well as what the tone type means, specifically in relation to the note type, will be discussed. For now, it is critical that one recalls

that tone literals must start and end with a `'/'` character to distinguish them from other literals.

3.1.4.6. Octave Literals

Octave literals are regulated as such: `'/' ['0' - '9'] '/'`. Once again, this literal is significant in the context of the note type and the note type's assignment. For now, it is critical that one recalls that octave literals must start and end with a `'/'` character to distinguish them from other literals.

3.1.4.7. Rhythm Literals

Rhythm literals require the following regex: `'/' (['s''e''q''h''w']['.']?) '/'`. Once again, this literal is significant in the context of the note type and the note type's assignment. For now, it is critical that one recalls that rhythm literals must start and end with a `'/'` character to distinguish them from other literals.

3.1.4.8. Note Literals

Note literals, this, meaning the literal that can be assigned / is of type note, exist in the C♭. In general, note literals look like the following: `( /C-/ /4/ /h/ )`. Observe that this looks like a tone literal, octave literal, and rhythm literal surrounded by parenthesis; this is not a coincidence. More on this in the following sections.

3.2. Manual Notation

The current section describes fonts and notation used in this manual. In this manual, sections are named in bold and numbered in dotted-decimal format for convenient referencing. Furthermore, nested sections generally refer to corresponding belonging and topical relevance. Throughout the manual, the `Courier New` font is used to refer to specific / explicit sequences of characters that will be used in the C♭ language, in respective scenarios. Double or single quotation marks around the `Courier New` text simply gives added emphasis on the actual spelling-out of each character itself.

The *italics* font is used to describe more generic syntactic categories. This is especially relevant when explanations benefit from more general references to concepts, and not explicitly the exact term used to describe a specific concept in Ocamllex.

3.3. Types

In the C♭ language, two main characteristics of any identifier are to be understood at all times — that is, the scope of the identifier and the type of the identifier. Type must be stated explicitly by the user upon declaration. (Although this will be elaborated upon in latter sections, it may be useful to learn that all variables used in a C♭ program must be declared in somewhat of a "declaration chunk" in the beginning of the program.) As a rule, identifiers must be declared exactly once before assignment or any other use, before any other non-declaration statements are made.

There are nine types supported by C♭ are as follows: void, integer, float, string, boolean, note, tone, octave, rhythm, and array. Note, tone, octave, and rhythm are novel data types in C♭. They are explained at great lengths in this section, along with array, which are a composite data type.

3.3.1. Primitive Types

Out of the nine types, eight — void, integer, float, string, boolean, note, tone, octave, and rhythm — are known as primitive types. Essentially, this is anything but arrays.

3.3.2. Note

In the C♭ language, every note is defined by its three attributes — tone, octave, and rhythm (the duration of the pitch). After declaration, during assignment, the value of each note's tone, octave, and rhythm can be explicitly stated together in / as the note literal. Upon declaration (see `note x;` below), the note takes on default values for tone, octave, and rhythm. Notes are a composite data type, in that they can hold more than one value — in this case, of different types. The following lines show examples of how notes are declared and assigned with a note literal in C♭:

```
note n;                    (: Declaring note n :)
n = (/C-/ /4/ /h/);        (: Assigning note n with note literal :)
```

3.3.2.1. Tone

There are 12 tones in a single Western musical octave, represented by a letter (C, D, E, F, G, A, B) and an accidental (♮ ♭ ♯). The 12 tones evenly divide an octave and are each separated by a half-step. The C♭ language obeys this convention. To represent a tone, letter names are modified by an accidental. The ♮ (natural symbol) indicates the natural pitch of the letter. The ♯ (sharp symbol) raises the natural letter pitch by a half-step, and the ♭ (flat symbol) lowers the natural letter pitch by a half-step.

The C♭ language utilizes capital letters A through G to represent tones, and "+" to represent a sharp, "–" to represent a flat. The absence of any accidental assumes a natural. A note is like a specific built-in C♭ struct data type. The tone of a note is a specific attribute of the note data type.

```
tone t;                    (: Declaring tone t :)
t = /C-/;                  (: Assigning tone t with tone literal :)
```

3.3.2.2. Octave

The octave value acknowledges which octave a tone belongs to, which is important because there are theoretically infinitely many notes for each of the 12 tones. The C♭ language defines the octave value of middle C as `4`. The octave of a note is a specific attribute of the note data type.

```
octave o;                  (: Declaring octave o :)
o = /4/;                   (: Assigning octave o with octave literal :)
```

3.3.2.3. Rhythm

The rhythm of a note is a specific attribute of the note data type. A note requires a rhythm value to indicate the amount of time a note is held. The C♭ language strays from conventional music theory in that it does not regard traditional time signatures in the same way. (Instead, the programmer may be liberal with the tempo. See sections describing the bplay function.) In C♭, users are able to assign the rhythm of a note from a finite set of possible rhythms.

Rhythm								
Number of Beats	0.25	0.5	0.75	1.0	1.5	2.0	3.0	4.0
Letter Syntax	s	e	e.	q	q.	h	h.	w

3.3.2.4. Rest

Rests are special notes whose rhythm attribute is specified in the same way as any note. The rest tone is represented as `/R/`. Moreover, the octave of a rest is `0` by convention, but this also does not affect the rest. A rest is simply the absence of a played tone for some beats, determined by the rhythm value. Rests mean silence.

3.3.2.5. Methods for Notes

Additionally, the note data-type in the C♭ language possesses built-in methods

which are shown below. These methods may be used to access or reassign attributes of a note. Let `n` be the default note — `note n = (/C-/ /4/ /h/);` — for which the following methods are executed to obtain the following return values:

Get Methods (accesses and returns)

Method	Return Value	Return Type
`n.tone()`	`C-`	*tone*
`n.octave()`	`4`	*octave*
`n.rhythm()`	`h`	*rhythm*

Set Methods (sets and returns set value)

Method	Return Value	Return Type
`n.tone(/B-/)`	`B-`	*tone*
`n.octave(/2/)`	`2`	*octave*
`n.rhythm(/q./)`	`q.`	*rhythm*

3.3.3. Array

The array data type is another composite/compound data type. It stores an ordered and indexed sequence of any number of objects, as long as all the objects are all of the same type. Depending on the type of the objects inside the array, the declaration of an array is different. For example, using an arbitrary literal `A`, the following are the only valid array declarations in C♭ — "`int[] A;`", "`bool[] A;`", "`float[] A;`", "`char[] A;`" "`string[] A;`", "`note[] A;`". Note that C♭ only supports one-dimensional arrays.

The size of the array, i.e. how many objects it can store, must be declared with `make(A, size)`, where `A` is the name of the array, and `size` is an `int` that represents the number of elements in the array.

Each array element can be accessed such as `A[index]`, where `A` is the name of the array, and `index` is an `int` that represents the index number of the element to be accessed. Arrays in C♭ are zero-based.

```
note n = (/C-/ /4/ /h/);
note[] arr;                 (: Declaring tone arr :)
arr = make(note, 10);       (: Allocating memory; arr hold 10 notes :)
arr[0] = n;         (: Assigning note n to the first element of arr :)
```

3.4. Expressions

3.4.1. Primary Expressions

Primary expressions consist of identifiers, constants, note expressions, parenthesized expressions, and function expressions.

3.4.1.1. Identifiers

An identifier (of the token *ID*) is a primary expression, and its type is specified by its declaration.

3.4.1.2. Constants

The literals mentioned in section 3.1.4 are primary expressions, which the grammar calls *LITERAL*, *FLIT, STRLIT, tlit, olit, rlit* and *notelit*.

3.4.1.2.1. Notes

A note literal is a primary expression that the grammar calls *notelit* and is of the form (*tlit olit rlit*).

3.4.1.3. (*expression*)

A function expression is a primary expression that the grammar calls *func_expr* and is of the form *ID*(*args_opt)*.

3.4.1.4. Functions

A function expression is a primary expression that the grammar calls *func_expr* and is of the form *ID*(*args_opt)*

3.4.2. Unary Operators

C♭ has the following unary operators: `-` and `!`. The negation operator `-` will negate the `int` or `float` that comes after it. The NOT operator `!` negates the `boolean` that comes after it.

3.4.3. Binary Operators

C♭ has the following binary operators: `+`, `-`, `*`, `/`, `==`, `!=`, `<`, `<=`, `>`, `>=`, `&&`, and `||`. The two operands of a binary operation must have the same type. Expressions

with binary operators have the form *expression* binop *expression*.

3.4.3.1. Arithmetic Operators

The multiplication operator * and the division operator have the same precedence and are all of higher precedence than the addition operator + and subtraction operator -. Arithmetic operators have higher precedence than relational operators and are left-associative.

3.4.3.2. Relational Operators

The relational operators <, >, <=, and >= return either `true` or `false`. Relational operators have higher precedence than equality operators and are left-associative.

3.4.3.3. Equality Operators

The equality operators == and != are analogous to the relational operators and return either `true` or `false`. Equality operators have higher precedence than boolean operators and are left-associative.

3.4.3.4. Boolean Operators

The `&&` operator returns `true` if both its operands are `true`, and `false` otherwise. The `||` operator returns `true` if one of its operands are `true`, and `false` otherwise. The `&&` operator has higher precedence than the `||` operator, which has higher precedence than the assignment operator.

3.4.4. Assignment Expressions

The assignment operator = groups right-to-left and requires an lvalue as its left operand. The value of an assignment expression is the value stored in the left operand after the assignment.

3.5. Program Structure

A C♭ programs essentially consists of a list of declarations, among which there must be a function declaration that declares a function named `main`, and `main` must return either an `int` or `void`. Any variable declared inside functions belong to the scope of that function, while

variables declared outside functions are treated as global variables. Rules regarding scopes are further discussed in section 9.

The syntax for both variable and function declarations are elaborated in the following section.

3.6. Declarations

3.6.1. Variable Declarations

Variable declarations are used to specify the interpretation given to each variable identifier. Variable declarations have the form

type_specifier declarator `;`

We will discuss both of these components in the following subsections.

3.6.1.1. Type Specifiers

The type specifiers include

`int`
`float`
`bool`
`string`
`note`

All of their corresponding types are discussed in detail in section 4.

In addition, to declare an array variable, the type specifier would have the form

element_typ `[]`

where element_typ is the data type of elements contained in the array.

3.6.1.2. Variable Declarator

The variable declarator has the form

identifier

where identifier is the programmer-defined name of the variable being declared.

Identifiers must belong to the regular set that belongs to the regular expression

`['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`

3.6.2. Function Declarations

Function declarations in C♭ follow the form

`def` return_type identifier `(` $\text{formals}_{\text{opt}}$ `)` `{` body_list `}`

where return_type specified the type of the function's return value (types that can be a return_type are listed in section 4.7); identifier is the programmer-defined name of the function; formals is the optional list of parameters passed into the function; body_list is a list of statements to be executed in the scope of this function.

3.7. Statements

3.7.1. Expression Statements

Most statements are expression statements, which have the form

expression `;`

Usually expression statements are assignments or function calls.

3.7.2. Conditional Statements

The two forms of the conditional statement are

`if (` expression `)` statement

`if (` expression `)` statement `else` statement

In both cases the expression is evaluated and if it is true, the first substatement is executed. In the second case, the second substatement is executed if the expression is false.

3.7.3. While Statements

The `while` statement has the form

`while (` expression `)` statement

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

3.7.4. For Statements

The `for` statement has the form

`for (` expression-1_{opt} `;` expression-2_{opt} `;` expression-3_{opt} `)` statement

The first expression specifies initialization for the loop; the second specifies a test, made before each iteration such that the loop is exited when the expression becomes false; the third expression typically specifies an incrementation which is performed after each iteration.

3.7.5. Return Statements

A function returns to its caller by means of the return statement, which has one of the forms

```
return ;
return expression ;
```

In the first case, no value is returned. In the second case, the value of the expression is returned to the caller of the function.

3.8. Scope Rules

Variables declared in a function are visible within that function. Variables can also be declared in conditional statements and have scope within that statement. Moreover, a variable overrides another variable with the same identifier if it has a smaller scope. A variable with wider scope can be accessed from a block with a lower scope level, given that its identifier is unique to that level.

The storage class is never explicitly state by the user; no keywords in the C♭ language pertain to storage class or scope of an identifier.

Elaborating first on storage classes, the break-down of storage classes and variable lifetimes in C♭ is extremely simple. A variable is either global to the entire program, or local to only the function in which it is declared, after its declaration. The location in the program of such a declaration immediately assumes this natural differentiation of scope, and storage classes are determined naturally to the identifier's initialization.

3.9. Built-in Functions

3.9.1. Print Functions

In addition to the standard print functions provided by microC (`printf`, `printbig`, etc.), C♭ also supports the printing of its special types, i.e. note, tone, octave and rhythm, through their respective print functions: `printn(note n)`, `print(tone t)`, `printo(octave o)` and `printr(rhythm r)`. Each of those functions will print the content of its parameter to standard output in the form of a string.

3.9.2. playnote

C♭ allows users to generate a playable MIDI file for a single note using `playnote(note n)`. The default tempo for `playnote` is 120 bpm.

3.9.3. bplaynote

In addition to `playnote`, C♭ also allows users to generate a MIDI file for a note with custom tempo such as `bplaynote(note n, int bpm)`, where `bpm` is the number of beats per minute specified by the user.

3.9.4. playtrack

Rather than a single note, `playtrack(note[] arr)` takes an array of notes as its parameter, and generates a MIDI file for the entire array. The default tempo for `playtrack` is also 120 bpm.
As of the writing of this documentation, there is a known bug with this function: for unknown reasons, `playtrack` can only generate MIDI files for note arrays with a hard-coded size of 23. We intend to fix this bug in the future.

3.9.5. tone, octave and rhythm

C♭ takes an OOP-like approach to manipulate the note type as well as its attributes.
Suppose variable `n` is of type note: the user can then access attributes of `n` using `n.tone()` (which returns a tone object), `n.octave()` (returns an octave object) and `n.rhythm()` (returns a rhythm object).
The user can also modify attributes of `n` using `n.tone(tone t)`, `n.octave(octave o)` and `n.rhythm(rhythm r)`. In this case, `tone`, `octave` and `rhythm` return nothing.

4. Project Plan

4.1. Planning, Specification, Development, and Testing

Our team met 2 times a week on average to check in with each other on project progress and to determine next steps. During our team meetings, we often worked together to solve any issues a team member might have had and also determined next steps in the project and split up tasks. We also met weekly with our TA Harry Choi to ask any questions we had at that point. Throughout the week, we messaged each other regarding any concerns that may have come up or to schedule one-off meetings with each other.
For development, we often paired up to gain an understanding of how to accomplish each task. Tests were primarily written prior to developing the corresponding features.

4.2. Style Guide

Our team aimed to write clean, organized Ocaml and C code. We generally used the following guidelines while writing our compiler:

1. Frequently commit.
2. Clearly indent.

3. Use descriptive variable/function names.
4. Keep lines under 100 characters.
5. All SAST types are named as the corresponding AST type with the addition of a capital ‘S’ at the beginning.
6. Simplify programs if possible.
7. Keep code modular and reusable.

4.3. Project Timeline

Date	**Tasks**
1/24/21	decide on type of language
1/30/21	finalize syntax
2/3/21	submit project proposal
2/24/21	complete LRM, parser
3/24/21	compile “Hello World”
3/31/21	begin integrating midi C-library to compiler
4/4/21	add C-like features to compiler
4/15/21	complete CFlatAPI
4/23/21	final project report, final project presentation, compiler for the language

4.4. Team Roles and Responsibilities

4.4.1. Team Roles

Team Member	**Role**
Jasmine Valera	Project Manager
Isabella Cho	Systems Architect
Katie Kim	Language Guru
Yvonne Chen	Tester

4.4.2. Responsibilities

Responsibilities were often split amongst more than one person, and we often used pair programming while developing our compiler. Below shows a breakdown of who is mainly responsible for each task, although everyone partook at least somewhat for most of the tasks.

Initial scanner, parser, LRM	Katie, Yvonne, Isabella, Jasmine
C-features	Isabella, Jasmine
Note type	Isabella, Jasmine
Note attributes and operators	Isabella
Arrays	Yvonne, Jasmine
Built-in function Integration	Jasmine
Testing/Demo	Katie
CFlatAPI	Katie
Final Presentation	Katie, Yvonne, Isabella
Final Report	Katie, Isabella, Yvonne, Jasmine

4.5. Software Development Environment

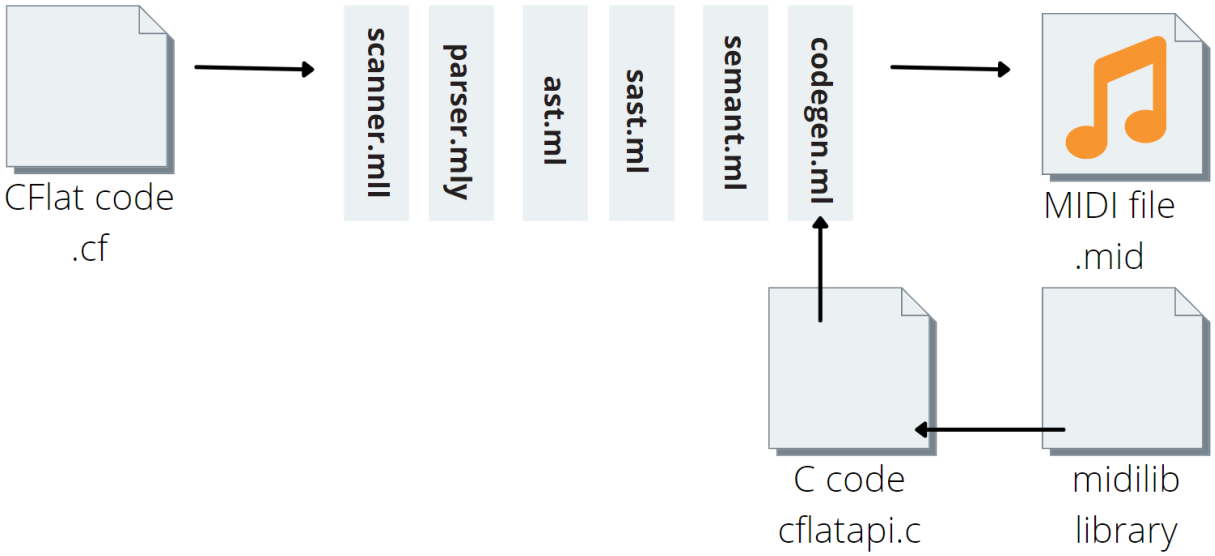
Libraries and Languages: OCaml version 4.05.0
OCaml LLVM version 10.0.0
OCamlyacc version 4.05.0
Ocamllex version 4.05.0
C & C libraries: stdio.h,
stdlib.h,
string.h,
MIDI Library (https://github.com/MarquisdeGeek/midilib)

4.6. Project Log

See Appendix 8.1 Project Log for all our GitHub commit messages that show the course of the project development.

5. Architectural Design

The CFlat compiler consists of 7 modules that work together to transform CFlat code into a binary file which, when run, can produce a MIDI file as needed. This flow is shown in the diagram below.



5.1. Scanner

The scanner takes in a C♭ program (which is essentially a series of ASCII characters) and translate them into tokens; those tokens will be used to identify keywords, operators, and a variety of other programming language components. If any characters in the code are detected to be illegal, lexing errors will be thrown. Spaces, newlines, and well as characters inside the commented blocks will be ignored.

5.2. Parser

The parser evaluates the tokens generated by the parser and, according to the C♭ grammar specified in our Language Reference Manual, converts them to an abstract syntax tree (AST) by matching tokens with AST nodes. If any mismatching (i.e. syntax error) is detected, a parser error will be thrown.

5.3. Semantics

The semantic checker (semant.ml) runs through the AST (as specified in ast.ml), checks for typing, and converts it to a semantically checked abstract syntax tree consisting of properly typed objects (as specified in sast.ml). This process is especially important for our note type

(which holds three attributes, each with a specified type), as well as the array (which can only hold elements of the same type). If a type error occurs, sement.ml will output an error message.

5.4. Code Generation

The code generator (codegen.ml) traverses the SAST, and generates the appropriate LLVM call for each SAST node to build the LLVM equivalent for C♭.
In addition to the standard functions of microC, our codegen.ml generates a series of standard C♭ functions; all of these functions are discussed in the Built-in Functions section of our Language Reference Manual. For some of these functions (namely playnote, bplaynote and playtrack), their implementation utilized a custom interface between LLVM and a third-party library, which will be described in more detail in the next section.

5.5. Linking, API and C Library

For our built-in functions that require the creation of MIDI files (namely playnote, bplaynote and playtrack), we implemented an API (named `cflatapi.c`) that were linked into codegen.ml through `L.declare_function`. This API imports a third-party C library named midilib (https://github.com/MarquisdeGeek/midilib), and utilizes many of its functions and structs for MIDI file manipulation.

All four team members work on the scanner, the parser, and the code generation. The AST and SAST were mostly implemented by Isabella and Jasmine, with the array components implemented by Yvonne. The MIDI library interface was mostly implemented by Katie.

6. Test Plan

6.1. Integration Tests

Our end-to-end integration tests can be found in the test/ directory. We each implemented modular tests to rigorously test the functionality of our language as we implemented it. Because we ran all of the previous tests each time we tested a new test, we ensured that none of the previously working features broke when we added new ones.

We have three different types of tests in our test suite. All of the tests that successfully produce output to standard output (mainly tests that call print, printb, printf, prints, printn) follow the naming pattern of using test-*.cf for the test program and then using the same name but different file extension of the form test-*.out for the expected output of the program. All of the tests that successfully produce a midi file as the output (i.e. tests that call playnote, bplaynote) follow the naming pattern of using midi-*.cf for the test program. The output should be a midi file with the same name as the test midi-*.mid. This file should be created in the CFlat_ directory. We have a matching midi-*.mid file with the same name as the test and the created midi file inside the tests/

directory that should match the output midi file that is generated by the midi test. All of the negative tests similarly used fail-*.cf for the test program that should fail and used fail-*.err instead of .out for the expected error message.

6.2. Testing Automation and Scripts

To run all of our tests, we modified the Makefile and the testall.sh script provided by Professor Edwards for the MicroC compiler. The script runs all of the midi-*cf, test-*.cf, and fail-*.cf programs sequentially, and compares it to the .mid, .out, .exe output files respectively. If the output of the program matches it's expected output file (whether that be a .mid, .out, or .exe file), it will print an OK after the test. If the output file doesn't match the expected output, it will print FAILED next to the test. The testall.log file in the CFlat_ directory contains a more detailed error message that can be used to debug. For the test-*.cf and fail-*.cf test programs that fail, a .diff file will be generated that will show the difference between the expected and the actual output of the program. For midi-*.cf test programs, the .diff file usually just states that they are different, which isn't very helpful. We have found it helpful to use LogicPro to listen for the difference. Any other Digital Audio Workstation (DAW) or midi sequencer can also be used to do this.

6.3. Sample Test Programs

6.3.1. Demo 1: letsdoit

This program creates the intro to the song "Let's Do It," by creating the notes necessary and creating an array of notes that will be passed into the built in function playtrack() to generate a midi file based on those notes. This program demonstrates declaring notes and an array of notes using our make function to allocate space for our not array. Then, it demonstrates a variety of tones, octave, and rhythm, that we can populate the attributes of our note with. This also demonstrates the use of built in functions for notes such as .raiseOctave() in order to increase the octave of a note in place, without having to create a whole new note. Then, the built in function playtrack() is called, which uses our cflatapi library to generate a Midi file that the programmer is then able to play. We made this test because it highlights both the declaration and assignment of our note and array and shows our play method functions.

```
int main()
{
    note[] letsdoit;
    note a_f; note b_f; note c; note d_f; note e_f; note e_flat;
    note f; note g; note c_up; note f_eighth; note e_eighth;
    note a_eighth; note b_eighth; note d_eighth; note c_eighth;
    note r;
    octave o;
```

```
int i;
note forNote;
string file;

file = "LetsDoIt";
letsdoit = make(note, 25);
o = /4/;

a_f = (/A-/ /4/ /q/); a_eighth = (/A-/ /4/ /e/);
b_f = (/B-/ /4/ /q/); b_eighth = (/B-/ /4/ /e/);
c = (/C/ /4/ /q/);    c_eighth = (/C/ /4/ /e/);
c_up = (/C/ /5/ /q/);
d_f = (/D-/ /4/ /q/); d_eighth = (/D-/ /4/ /e/);
e_f = (/E-/ /4/ /q/); e_eighth = (/E-/ /4/ /e/);
f = (/F/ /4/ /q/ ); f_eighth = (/F/ /4/ /e/);
g = (/G/ /4/ /q/);
r = (/R/ /4/ /q/);
letsdoit[0] = f_eighth;
letsdoit[1] = e_eighth;
letsdoit[2] = f_eighth;
letsdoit[3] = a_eighth;
letsdoit[4] = f;
letsdoit[5] = e_f;
letsdoit[6] = c_eighth;
b_eighth.octave(/3/);
letsdoit[7] = b_eighth;
a_eighth.octave(/3/);
letsdoit[8] = a_eighth;
letsdoit[9] = b_eighth;
letsdoit[10] = c_eighth;
letsdoit[11] = d_eighth;
letsdoit[12] = e_f;
letsdoit[13] = f_eighth;
letsdoit[14] = e_eighth;
letsdoit[15] = f;
```

```
    letsdoit[16] = c_up;
    letsdoit[17] = c_up;
    letsdoit[18] = r;
    letsdoit[19] = a_f;
    f.octave(/5/);
    letsdoit[20] = f;
    letsdoit[21] = f;
    letsdoit[22] = r;

    playtrack(letsdoit, "Letsdoit");
    return 0;
}
```

6.3.2. Demo 2: letsdoit_modify

This function is source code 1 along with a control for loop added to it which will loop through our created array of notes and increment the octave of each note by two. This will generate a midi file called midi-demo2.mid which plays the melody from source code 1 two octaves higher. In this way, Programmers can use our control loops to more powerfully act upon our note arrays.

```
int main()
{
    note[] letsdoit;
    note a_f; note b_f; note c; note d_f; note e_f; note e_flat;
    note f; note g; note c_up; note f_eighth; note e_eighth;
    note a_eighth; note b_eighth; note d_eighth; note c_eighth;
    note r;
    octave o;
    int i;
    note forNote;
    string file;

    file = "LetsDoItOctave";
    letsdoit = make(note, 25);
    o = /4/;
```

```
a_f = (/A-/ /4/ /q/); a_eighth = (/A-/ /4/ /e/);
b_f = (/B-/ /4/ /q/); b_eighth = (/B-/ /4/ /e/);
c = (/C/ /4/ /q/);    c_eighth = (/C/ /4/ /e/);
c_up = (/C/ /5/ /q/);
d_f = (/D-/ /4/ /q/); d_eighth = (/D-/ /4/ /e/);
e_f = (/E-/ /4/ /q/); e_eighth = (/E-/ /4/ /e/);
f = (/F/ /4/ /q/ ); f_eighth = (/F/ /4/ /e/);
g = (/G/ /4/ /q/);
r = (/R/ /4/ /q/);
letsdoit[0] = f_eighth;
letsdoit[1] = e_eighth;
letsdoit[2] = f_eighth;
letsdoit[3] = a_eighth;
letsdoit[4] = f;
letsdoit[5] = e_f;
letsdoit[6] = c_eighth;
b_eighth.octave(/3/);
letsdoit[7] = b_eighth;
a_eighth.octave(/3/);
letsdoit[8] = a_eighth;
letsdoit[9] = b_eighth;
letsdoit[10] = c_eighth;
letsdoit[11] = d_eighth;
letsdoit[12] = e_f;
letsdoit[13] = f_eighth;
letsdoit[14] = e_eighth;
letsdoit[15] = f;
letsdoit[16] = c_up;
letsdoit[17] = c_up;
letsdoit[18] = r;
letsdoit[19] = a_f;
f.octave(/5/);
letsdoit[20] = f;
letsdoit[21] = f;
letsdoit[22] = r;
```

```
    for (i = 0 ; i < 23 ; i = i + 1) {
        forNote = letsdoit[i];
        forNote = forNote.raiseOctave(2);
        letsdoit[i] = forNote;
    }
    playtrack(letsdoit, file);
    return 0;
}
```

6.4. Testing: Division of Labor

The source was written by Katie. However, the end to end integration tests described in 6.1 were written by everyone in the team as they implemented the functions they were testing. As the members of the team developed each feature, tests were written and distributed.

7. Lessons Learned

Yvonne Chen:

1. Functional programming is really cool. As particular as it is, I feel like by suffering through OCaml I learned a lot about programming in general.
2. I (re)learned the importance of time management. In this project, we encountered lots of expected and unexpected roadblocks: some of them forced us to take a completely new approach, some of them left us no choice but to wipe the slate clean and start over, and some of them just took way longer than we ever anticipated to resolve. Basically, when you have no idea how to solve a problem, you should just leave as much time for trial and error as possible.
3. Sometimes you need to understand the end product and work backwards. Rather than spending way too much time going back and forth on syntax and features, the VERY first thing we should have done was to thoroughly learn what a MIDI is. This mistake led us to make some very significant feature changes about 1.5 weeks from the due date.
4. TA Harry is a gem, and sometimes (oftentimes) getting help is the most productive option, even when you feel like you're too scared about not knowing how things work.

Isabella Cho:

Before this project, I thought I knew a thing or two about working with other people on a group project. Everything I thought I knew was debunked, and relearned this semester. I thought I was a patient person. To this, well, I suppose I had never written a whole compiler from beginning to end with three other independently minded individuals before. In addition to all the technical

skills I gained from this project, I gained so much more from the experience of having to make compromises when opinions arise, communicate clearly so as to avoid future confusions, maintain constant communication, realize and recognize my own behavior and tendencies when working and making mistakes and fixing mistakes under high stress. I certainly feel more prepared to know how, even the best of teams, which I do believe our team was regarding our variety of skills and strengths, good-intentions, and hard-working / curious / thorough attitudes. However, this is not to overshadow the fact that I gained more in technical skill than I could have imagined. This was not a project one could simply… "fake it 'til you make it"—to completion. Also, the attention to detail and consistent billion part understanding and thinking required to implement a program from scanner to codegen, then write tests for, then fix, then communicate with team members about branches and merge conflicts with, was the most laborious, but also rewarding experience ever. I am satisfied with how far I have come, given that codegen and lots of OCaml used to scare me. I now feel like an LLVM.moe master. I feel comfortable taking risks while coding in OCaml now, and then getting taught another lesson about being careful and intentional to every detail of every value / type and function. At the end of the day, however, the most important things I have learned from this project are: 1) When your teammates lose momentum on the project timeline, it is not time to take a breather, rather, you must pull double the weight. 2) I should reach out to TAs and the professor for help much earlier than I typically have gotten into the habit of doing. 3) Trust the advice that is given to you, and look for correctness in the details. This is also the advice I would like to give to figure teams. Don't not go to the TA because you don't know what is going on / what to do / how to start; rather, go to the TA because you don't know. Don't let yourself take breaks from working PLT throughout the semester. Never stop. PLT is life. You signed up for this, so just enjoy it and grow to love the .moe. Also, TA Harry is the best human ever, and Professor Edwards is not only a brilliant human, but actually won't bite your head off during 1:1's during office hours, and is ACTUALLY HILARIOUS. Finally, when the professor says to do something more than twice, definitely, definitely, definitely do it. Don't be stubborn like me.

Katherine Kim:

Sometimes you spend a lot of time writing up C functions that would make Prof Jae cry tears of joy, and they don't get used. And that's just life.

In all seriousness, this project really challenged my communication and management skills. I learned that I get the most frustrated with others when working on a team when I myself don't have it all together or know what I'm doing. And when I spent nearly 2 whole days banging my head on arrays, all I could hear was Professor Edwards respond in a piazza post "Any idiot can spend a bunch of time putting in effort, but we're looking for mastery of the subject." (Source - paraphrased from some random piazza post). Some harsh truth that really simmered in my head as we got closer to the deadline.

I think our group did a decent job of not sticking to our assigned job label and just filling in the role/task that was needed. While I was the "language guru", I definitely got a taste of what being the manager was at times and it was difficult trying to delegate tasks when I wasn't sure what exactly needed to get done.

I'm really glad I got to take this class at this moment in my school/work career. Our group worked decently well but I can definitely see myself encountering many of the issues our team did like miscommunication, lack of communication, not knowing the next steps, not knowing who/how to ask for help etc. and I feel better equipped to deal with these issues the next time I work on a major group project - which could very well be in a weeks.
My advice to future groups is to spend less time talking and planning your language and dive more into the programming ASAP. We spent a lot of time getting into the logistics of our vision for our language and in the end got around to only implementing maybe 1/3 of it.

Jasmine Valera:

Through this project, I got to learn about functional programming and came to appreciate how useful it is. I also thought it was really eye-opening to see the inner-workings of a compiler, which was entirely a blackbox to me prior to this project. I learned a lot about the importance of dividing up tasks into smaller, more manageable tasks. My best advice is to dive into coding as soon as possible to get a feel for OCaml and generating LLVM. Don't let the complexity of this project intimidate you and keep you from moving forward!

8. Appendix

8.1. Project log

commit 67735eeab6293a084645a263938c2101d1fc5b56
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Tue Apr 27 04:49:18 2021 -0400

    Katies latest changes
    with demo as midi-demo and midi-demo2

commit fc547b94e90c4587c8899d48f0a872527d88c669
Merge: 47c26c2 d3ad87d
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Tue Apr 27 02:46:42 2021 -0400

    Committed izzi's changes

commit 47c26c226e8a3df33718aaf36e5371d00c3e5a82
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 21:04:44 2021 -0400

play arraygit add semant.ml playarr.mid codegen.ml cflatapi.h cflatapi.c tests/midi-play_arr.cf tests/test-arr_make_note.cf

commit 069d49fb32de870d9a66ae016a420e25154fa63c
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 20:57:18 2021 -0400

test midi play arr file

commit d3ad87d59db072f36a7dc584c32e3d5252247f30
Author: isabellacho <isc2120@columbia.edu>
Date: Mon Apr 26 20:33:04 2021 -0400

nothing new. just trying new things for doing raiseTone() based on what edwards said on piazza :(sad

commit 20d38e8ed65f8ea2e8801ccc164e62704c0711c0
Author: isabellacho <isc2120@columbia.edu>
Date: Mon Apr 26 14:44:59 2021 -0400

IMPLEMENTED RAISE AND LOWER OCTAVE

commit 360e48c2a7d9057549300e4d5f9628ddf7b2a5ab
Merge: 52706f1 a09cd6b
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 12:57:19 2021 -0400

Merge branch 'fudgedRaise' of https://github.com/katieskim/CFlat into playNote

commit 52706f135ce33ae88ae3817b80128b857a6144f3
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 12:57:05 2021 -0400

generate code for int arrays working

commit a09cd6b1de4d0f90b10c31a14ba1013f7d91b99f
Merge: a2fda98 6ba258e
Author: isabellacho <isc2120@columbia.edu>
Date: Mon Apr 26 12:44:20 2021 -0400

resolved merge conflicts between playNote and izzibranch

commit a2fda98a7ee7770e360a4cd542e3212aff650388
Author: isabellacho <isc2120@columbia.edu>
Date: Mon Apr 26 12:12:23 2021 -0400

    GREEN CHANGES IN OUR NOTES GOOGLE DOC HAVE BEEN IMPLEMENTED

commit 6ba258ed8613b0f7b1070f4011cd6fced5f0b2e9
Merge: fa668bf 3d7ff09
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 10:58:48 2021 -0400

    illegal assn semant error

commit fa668bffe73a6fc7e67f51143ef793b746d90384
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 10:45:56 2021 -0400

    semant

commit 6c362d6d96901966928f38d857d9ac0795f3b40a
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 10:44:22 2021 -0400

    passing play tests except arr

commit 9f0b12d3d7575230afb1e2d64eb5c6fd60570386
Merge: b708cb8 24080a2
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 10:39:52 2021 -0400

    merge with katie

commit 24080a2f54fa718fc7456a5d3e5ae86c363cff81
Merge: 98610d0 cece230
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Mon Apr 26 10:36:48 2021 -0400

    fixed the tests

commit 3d7ff09d9f889b74a2a1386d81094f09e40a328e
Author: YvonneChenCS <jc5349@columbia.edu>
Date: Mon Apr 26 10:33:24 2021 -0400

Added array assign and access for everything except codegen

commit b708cb836e3782f2604b4500e2c0940b9f4e13b9
Merge: cece230 534840d
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 09:43:57 2021 -0400

merge midi file name changes

commit cece230568abc191dfdd435d0bf85e00d640f536
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 09:39:39 2021 -0400

makearray

commit 534840d5f1c9ccbd0d495fb79900d40f5715f4d0
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 08:48:01 2021 -0400

add to strlit

commit c8de91ed348c7c5c1841d2d9e71ee5a4c0a0a8ee
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 08:00:57 2021 -0400

add midi filenames to playnote and bplaynote

commit 9ae2499c804231bd4d79f70fe6cd781b703d5763
Merge: c62aa19 98610d0
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 07:35:06 2021 -0400

Merge branch 'katiearr' of https://github.com/katieskim/CFlat into jvcflatapi

commit 967d5edfa53ebd741b9bc9cc2d17c356ca0c43b6
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 07:25:59 2021 -0400

committing to branch

commit 2f273c34b24fd5f11be041ab363b22434408edfb

Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 26 07:23:35 2021 -0400

    stack overflow error

commit 98610d072fcdf6df0ef029c74aff52d31344e152
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Mon Apr 26 01:07:37 2021 -0400

    Testing shell script now tests midifiles

commit 999868faeeec858b47c9eab1cb1355bb78d70aa9
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 21:58:46 2021 -0400

    arr tests failing but running

commit 1b3d6eb9233a1c0be9397b0296e639b583ceaf6d
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 21:10:53 2021 -0400

    clean .s, .ll, etc

commit 6b1eda30e52b588f3a730629a7058d3b6cbca05b
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 21:00:56 2021 -0400

    Added functions to test is_tone_equal and is_note_equal

commit e01db91e5c72a0063f87cb9ead54a0560947ddea
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 19:40:10 2021 -0400

    Raise/lower tone and octave cflatapi functions

commit c62aa191688b7845c24d5a8a8a49bcad6f0df01f
Merge: 09a533a a8a690d
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Sun Apr 25 19:31:09 2021 -0400

    Merge pull request #12 from katieskim/playNote

Play note

commit a8a690d30003fe0aa1ad4db7a2fe93b49f351770
Merge: 4da8acb 09a533a
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:56:48 2021 -0400

resolve merge conflicts with set/get attrs

commit 09a533a23d0da977e63b18557f0d047506136585
Merge: 6582834 7a39f41
Author: izzicho <isabella.cho@columbia.edu>
Date: Sun Apr 25 15:50:17 2021 -0400

Merge pull request #11 from katieskim/getandset0

ALL THE .GET AND .SET METHODS WORK PERFECTTTTTTTT see my new test cal…

commit 4da8acbfe9a7e85df5235e2b5a4c6530ab6b3a85
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:49:23 2021 -0400

not passing array tests yet

commit 64f1bea782fb2b006d025d85cfa7548d38d1ea7e
Merge: 5574d69 944c9bd
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:46:40 2021 -0400

merge arr tests and cflatapi changes

commit 5574d69e7f9121ec09d8516012a417838fcba890
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:43:08 2021 -0400

delete cflatapi files

commit 7a39f414957222d89e6813f9332030fb5763afb8
Author: isabellacho <isc2120@columbia.edu>
Date: Sun Apr 25 15:41:19 2021 -0400

ALL THE .GET AND .SET METHODS WORK PERFECTTTTTTTT see my new test called test-noteatt3_getset.

commit 6e72a546ce81b2e019c1f8067b8803cd112e9ffc
Merge: 7c7dd66 6582834
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:37:06 2021 -0400

Merge branch 'main' of https://github.com/katieskim/CFlat into playNote

commit 944c9bdc3244fdbcfe1adc1f39108fd460b749d6
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 15:31:55 2021 -0400

Reverted change to test-playnote

commit 4a1b881b6808bf89111d1dbfea890b5c10ce0dda
Merge: 106c4be 7c7dd66
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 15:30:07 2021 -0400

trying to merge

commit 106c4be640ab5467d19531c96fdc91e743ac8bf9
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 15:27:30 2021 -0400

added filename

commit 7c7dd66160520eb33a6af25e6605fe49a2caecdb
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 15:08:05 2021 -0400

fix primitive type bugs

commit de90f759ad33969cbd9e8d2c2b75bd82349cc80c
Merge: 9130e67 4faeeb1
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 14:05:28 2021 -0400

adding tests

commit 4faeeb1cda2051e3b4991a6533b8d005c90977e9
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 09:25:26 2021 -0400

    add array type to codegen

commit 6b3a01743df613c274446fbe3f3446d06972e810
Merge: 99765b9 a3b2972
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 07:44:10 2021 -0400

    Merge branch 'yvarr' of https://github.com/katieskim/CFlat into playNote

commit a3b297294193a368a657e06f63600e0432182680
Author: YvonneChenCS <jc5349@columbia.edu>
Date: Sun Apr 25 02:01:06 2021 -0400

    Modified scanner to ast for array

commit 99765b9d03a0cb6ade8d9829ee1ab093634e5dfe
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 01:33:58 2021 -0400

    bplaynote passes

commit c71e27d0d38abf2cbaa1255dca3a1483404dfc5a
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 00:58:21 2021 -0400

    add bplaynote tests

commit 30f3f5bb9f80f8660498480a15e1fa4d4b4476d3
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sun Apr 25 00:57:39 2021 -0400

    add comma to scanner

commit 9130e6754a3db2d9be001a3c14fb07b989e2c49f
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Sun Apr 25 00:56:50 2021 -0400

    Added array tests

commit 6582834f0dba6e884a16e36f9c467f392bbeef07
Merge: f8c5081 9aeb577
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Sun Apr 25 00:51:13 2021 -0400

    Merge pull request #10 from katieskim/playNote

    Play note

commit 9aeb57712bc4e3da4bd10a4ff72f390e54dedcaa
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 23:21:27 2021 -0400

    fix endian bug

commit bad547df1391c841b9c67903c23f4b195c123c91
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 23:10:59 2021 -0400

    test midi

commit e902d60ef0bc688904bd971777bc64db00f536be
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 22:12:52 2021 -0400

    swapped cflatapi for cflatfile

commit 4e1850faa37e9284da6a60065548173e45df059a
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 21:46:40 2021 -0400

    test hellonote.mid

commit 5704b0a702727b18a03e231e27e5fcc1a75c9ba7
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 20:02:45 2021 -0400

    generate midi file with test-playnote

commit 16d5d948a24139dcf515b4203548881d77379d59
Author: jvalera174 <jasmine.valera620@gmail.com>

Date: Sat Apr 24 19:33:26 2021 -0400

    note passed to playnote

commit 429fcb341b5474931d4de930be737e4605bbc8bf
Merge: f8f620b 21bc8f6
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Sat Apr 24 16:19:50 2021 -0400

    merge cflatapi

commit 21bc8f658995ed50a6ba22154b3192c6e44b4df2
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 23:21:20 2021 -0400

    Github is making me commit again

commit f361ca10f667950e9a747a4b73a26e62f2646e17
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 23:20:33 2021 -0400

    cflatapi test

commit 89d40354f6881633b4210089095563ce820dd1b6
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 21:56:43 2021 -0400

    Added instrument functionality, and created header file for cflatapi

commit 338bd0e013d50a744d8fdbd70c2afcfb8971751a
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 20:03:45 2021 -0400

    User can play multiple tracks at oncegit status

commit 1d2b0371ffdf6c249aefa16ef004f5e77b269d9b
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 19:23:42 2021 -0400

    Can play two tracks

commit f8c5081249de329ee43d54d538b4a42bbae50c46

Merge: 677ca11 9c0b338
Author: Katie Kim <47716319+katieskim@users.noreply.github.com>
Date: Fri Apr 23 16:22:52 2021 -0400

    Merge pull request #9 from katieskim/cflatapi

    play_note, bplay_note, play_arr, bplay_arr, add_note. Currently working on adding tracks.

commit f8f620b5152c49d783f8f8a109476342729547ea
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 16:13:29 2021 -0400

    print note (actually)

commit 8ab9dcc938f7f1dca2781c3c552377afa41d2232
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 16:07:19 2021 -0400

    printing tone of note

commit 78d27ebe577f15275819579c7760175e7ff70e5b
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 15:28:38 2021 -0400

    resolved merge conflicts

commit eaf5a0d121da5f81e4faf4517634901e958fbb44
Merge: f3116b9 c9312e1
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 15:16:31 2021 -0400

    merged changes

commit f3116b9e014e8af8bf45af22b7cb57fea06166c2
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 14:41:47 2021 -0400

    linked midi files

commit 663ed869ba3645b6710f53b0db5406aa08ad6c81
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 14:41:08 2021 -0400

linked midi files

commit 852bc0109ab8916f56a210a3558afd2df5c193dd
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 23 14:37:40 2021 -0400

linked midi files

commit c9312e159e7b97e73e16b356702e0480397ccaa7
Author: isabellacho <isc2120@columbia.edu>
Date: Fri Apr 23 14:23:27 2021 -0400

i got rid of the warning and tried to make printn(n) work, but i get a segfault in the SCall of printn :(-izzi<3

commit 9c0b338b94ac721074af7acbc2f1b9672b629106
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 23 13:42:20 2021 -0400

Added bplaynote and bplayarr functionality

commit f62c896874bc37e5ee727ab32e18ef5a37a68363
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Thu Apr 22 20:32:33 2021 -0400

attr comments in semant

commit 6f4a79ac284656a89f8f5b269328b84ceb8f5172
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Tue Apr 20 00:25:46 2021 -0400

attempt note attributes

commit 843905e3c049ecb31e8e423dfce604a7822e0cf8
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 19 10:37:09 2021 -0400

remove build files

commit 6286fc62f35f26b732eb78e321af32a8022c0b6d
Merge: 3ec48c2 286b31c

Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Mon Apr 19 10:34:41 2021 -0400

    remove .DS_Store

commit 3ec48c2358d5617ac2e0051a18b80f8013128626
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Tue Apr 13 16:39:33 2021 -0400

    print tone of note

commit 286b31c7be3f4cba5f3f7864db4261519bfe37f6
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Tue Apr 13 16:21:58 2021 -0400

    HELLLO FIRIENND GLLLL

commit e3b1ddb25c8f31d0b1eb74716f000fe2892c005f
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Tue Apr 13 16:13:47 2021 -0400

    note struct allocated

commit 9c1db58f82ba185c644c58be87b60ad5b294e605
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Mon Apr 12 18:41:45 2021 -0400

    WHILE LOOP NOW WORKSSS AYOOOOOOO and both rests and accidentals work

commit 08cfea800eea496b5f5bac3fb6ee99e31216c58f
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Mon Apr 12 18:12:13 2021 -0400

    Fixed the accidental issue

commit defb659dd6c2263f8d44df46cf7efeb6134ef2a7
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Mon Apr 12 17:34:44 2021 -0400

    Most recent push. Currently having error playing accidentals and need to fix forloop condition

commit db98ae8a9cb3d7b806c151971b9f4de08334345c
Author: isabellacho <isc2120@columbia.edu>
Date: Mon Apr 12 00:38:51 2021 -0400

    AT LEAST WE GOT TONE AND OCTAVE AND RHYTHM! EVEN THO WE STILL STRUGGING WITH STRUCT NAMED BLAH BLAH STUFF STILL

commit 815492b8bb6b817b9001d8de2b93f8b0a474f1fe
Author: isabellacho <isc2120@columbia.edu>
Date: Fri Apr 9 22:02:54 2021 -0400

    named struct for note

commit afdcd8e949d1ba27a2d311fcd2410c57e6f83fb5
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 9 20:12:48 2021 -0400

    add struct c files

commit 782faec14aae92326f721d50d93f1c4056ed1e67
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Fri Apr 9 11:48:20 2021 -0400

    attempt note struct

commit 5bc0d4e084db00ba7733076f75c806e6aeb2eb64
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 9 02:52:51 2021 -0400

    Added input/output info about each function in cflatfile.c

commit a5dfe3f02a376bfa2d3cdd0409fd41516c68b73c
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Fri Apr 9 02:02:48 2021 -0400

    CAN PLAY AN ARRAY OF MUSIC NOW git add .git add .

commit b012c281cb9e386956c5a8966aba0eac39d2ac79
Author: isabellacho <isc2120@columbia.edu>
Date: Thu Apr 8 16:58:19 2021 -0400

    added note changes up to semant

commit 546be3d938807b83146221669c7bbc558459d94e
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Apr 7 22:22:41 2021 -0400

    Pushing midifile stuff in katiefile

commit bd08c64e36ea57f25b513d80f42988d62ba86f7c
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Apr 7 22:20:22 2021 -0400

    etest test

commit 2d54b157cb14c3375d945e699c51ffbe5b5f11d1
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Apr 7 13:13:15 2021 -0400

    CFlat API play_note() method

commit 677ca110e7fe4c718bbf8cd86876b5ea9b213e67
Author: isabellacho <isc2120@columbia.edu>
Date: Sun Apr 4 23:01:37 2021 -0400

    added {} to our tests. it worksgit status

commit e1c6e63bd4ac1795349ee2662266940eb3a0ecd7
Merge: 2e93429 c4e61fe
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Sun Apr 4 22:45:41 2021 -0400

    Merge pull request #8 from katieskim/izzi0

    Izzi0

commit c4e61fe8b9caf7bffdfd2c1eb71f535cc5eedeb9
Author: isabellacho <isc2120@columbia.edu>
Date: Sun Apr 4 22:42:56 2021 -0400

    jas and izzi implemented expressions and statements and made tests! and printed a string

commit 6429e791ab859704b965bc2b87a60a9c4533cfdf

Author: Isabella Cho <isabellacho@dyn-129-236-174-212.dyn.columbia.edu>
Date: Fri Apr 2 12:22:19 2021 -0400

    tried adding goodwin's stuff and making it work but make mfc120 has errors with midiinfo.h not linking?

commit 2e93429431b915b6909e3eb40a664528ad8862e2
Merge: 50869b3 011acc4
Author: izzicho <isabella.cho@columbia.edu>
Date: Tue Mar 23 23:45:08 2021 -0400

    Merge pull request #7 from katieskim/izzijasnote0

    printed (of note

commit 011acc465f056fa8d9fcaa4ed9d796a83e8280b1
Author: Isabella Cho <isabellacho@dyn-129-236-174-132.dyn.columbia.edu>
Date: Tue Mar 23 23:44:31 2021 -0400

    printed (of note

commit 50869b31d4f7306d62cf27378aeb3acd29d8802b
Merge: b5c4663 88a8ac0
Author: izzicho <isabella.cho@columbia.edu>
Date: Tue Mar 23 15:57:15 2021 -0400

    Merge pull request #6 from katieskim/trypointer0

    after harry's oh thingy

commit 88a8ac0b8c5fae44fa8b82d41c7a112feb1da935
Author: Isabella Cho <isabellacho@dyn-129-236-174-132.dyn.columbia.edu>
Date: Tue Mar 23 15:56:24 2021 -0400

    after harry's oh thingy

commit b5c4663a8ce5179056f2ecc984d4a16997535f89
Merge: ef5d533 c9ca171
Author: izzicho <isabella.cho@columbia.edu>
Date: Mon Mar 22 22:02:31 2021 -0400

    Merge pull request #5 from katieskim/printfloattest0

added printfloat test stupid

commit c9ca171677753cff57efbb538cbff3da2ef94048
Author: Isabella Cho <isabellacho@dyn-160-39-173-46.dyn.columbia.edu>
Date: Mon Mar 22 21:59:25 2021 -0400

added printfloat test stupid

commit ef5d533bec21461bb0fdd0a2c2c57640028df072
Merge: abd647c f830f1d
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Mon Mar 22 20:50:36 2021 -0400

Merge pull request #4 from katieskim/failtest0

make a fail test work

commit f830f1d8a8164d5d348a54ca5ceeaa903729624f
Author: Isabella Cho <isabellacho@dyn-160-39-173-46.dyn.columbia.edu>
Date: Mon Mar 22 20:44:29 2021 -0400

make a fail test work

commit abd647c5f89113ca95c57484d05b572deefd66c5
Merge: b7a85f2 e9bcf71
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Sun Mar 21 16:00:51 2021 -0400

Merge pull request #3 from katieskim/stripped_0

Stripped 0

commit e9bcf71320d78a25946c33a75de4f4b762208462
Author: Isabella Cho <isabellacho@dyn-160-39-173-46.dyn.columbia.edu>
Date: Sun Mar 21 15:59:12 2021 -0400

this is stripped CFlat_ is stripped if you make. it should be OK OK but fail tests should fail inappropriately bc they're not for .cf and they're still just for .mc but it's okay u can work on it

commit db97912b93ea8af6f7c8e129f2a1c96038ee0e27
Author: Isabella Cho <isabellacho@dyn-160-39-173-46.dyn.columbia.edu>

Date: Sun Mar 21 14:15:50 2021 -0400

    added CFlat_ folder that is a duplicate of the og og og squeaky clean microc. next we will strip it to make our hello world and hopefully build off from there. we only want stuff we UNDERSTAND in CFlat_

commit b7a85f273f4029edcd13dd718f0a70dee62788c7
Merge: 6817212 6b8804c
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Thu Mar 18 23:45:06 2021 -0400

    Merge pull request #2 from katieskim/ast

    Added some basic note features to ast, scanner, parser

commit 6b8804c1eebf6f8ceabcf9816de94a729b6726e6
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Thu Mar 18 23:44:18 2021 -0400

    Added some basic note features to ast, scanner, parser

commit 6817212de9dc1eb01aaa407bb82cfaf124e6bb17
Merge: 8fb5f15 db70bc9
Author: jvalera174 <46391086+jvalera174@users.noreply.github.com>
Date: Thu Mar 18 23:27:59 2021 -0400

    Merge pull request #1 from katieskim/ast

    Noahed the whole thing

commit db70bc97c1b8541c0497eb12d47ea7d28f27f774
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Thu Mar 18 23:26:51 2021 -0400

    Noahed the whole thing

commit 8fb5f15df5fac974514db9d963171d49e01656cd
Author: jvalera174 <jasmine.valera620@gmail.com>
Date: Thu Mar 18 20:19:28 2021 -0400

    add microc-starter folder

commit 585e87a3b2681790af07034dc39f77f66e8ff5d6
Author: Isabella Cho <isabellacho@dyn-129-236-174-159.dyn.columbia.edu>
Date: Sun Mar 14 10:34:22 2021 -0400

    first commit after meeting harry! time to change the 41 reduce errors time to throw expr into stmts

commit 5a283977ea3c117b3360549377b4a3a343ddbd3b
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 23:51:27 2021 -0500

    AYYYY reduced all reduce/reduce and shift/reduce conflicts

commit 0cee10d60fe51244f3ee762984d215fca002a44e
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 23:38:04 2021 -0500

    Reduced all shift/reduce errors

commit e83324ed3517b8bbb806acfa6938932a9ce7a20e
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 23:08:33 2021 -0500

    parser_save

commit 10302015c4567483c3a4958e14db1bb213baa926
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 23:06:24 2021 -0500

    ahhhhhh lets reduce some shift/reduce

commit e9b784dedae097d500ca7e7b7c07090a083c59f5
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 20:06:38 2021 -0500

    Fixed some syntax errors, hit shift/reduce errors

commit 59cd1eb901a5c7736e08b6208ceebee948a91077
Author: Isabella Cho <isabellacho@dyn-129-236-174-173.dyn.columbia.edu>
Date: Wed Feb 24 15:34:36 2021 -0500

Prettied up the parser and finished writing out all of me and Yvonne's parts up to right before stmt! Also prettied up the scanner a little.

commit da65d7d8a4a25e6fcc58d273a131f272e4421bc1
Author: Isabella Cho <isabellacho@dyn-129-236-174-173.dyn.columbia.edu>
Date: Wed Feb 24 14:20:30 2021 -0500

Uncommented the STRLIT and CHARLIT from the scanner.

commit 4af892b649b51f9a72da48ecc1388d4f4cce7f12
Author: Katie Kim <katherine.kim@columbia.edu>
Date: Wed Feb 24 02:31:05 2021 -0500

Restructured fils

commit 3f796bedf9cb77933855fb147c44df29cba4d361
Author: Katie Kim <47716319+katieskim@users.noreply.github.com>
Date: Wed Feb 24 02:24:39 2021 -0500

Uploaded CFlat project folder to Github.

commit ce7cbdcc7dd313b976ca2bf8947533c7152b1d09
Author: Katie Kim <47716319+katieskim@users.noreply.github.com>
Date: Wed Feb 24 02:20:05 2021 -0500

Initial commit

8.2. cflat.ml

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

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
```

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

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

8.3. scanner.mll

```
(* Ocamllex scanner for CFlat *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+
let esc    = '\\' ['\\' ''' '"' 'n' 'r' 't']
let ascii  = ([' '-'!' '#'-'[' ']'-'~'])

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "(:"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRACK }
| ']'      { RBRACK }                   (* arrays *)
| '{'      { LBRACE }
| '}'      { RBRACE }
```

```
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| "<"       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "&&"      { AND }
| "||"      { OR }
| "!"       { NOT }
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "make"  { MAKE }
| "return" { RETURN }
| "note"    { NOTE }
| "tone"    { TONE }
| "octave" { OCTAVE }
| "rhythm" { RHYTHM }
| ".tone()"    { TONEACCESS }
| ".octave()" { OCTAVEACCESS }
| ".rhythm()" { RHYTHMACCESS }
| ".tone"      { TONESET }
| ".octave"    { OCTAVESET }
| ".rhythm"    { RHYTHMSET }
| ".raiseTone"    { TONERAISE }
| ".raiseOctave"  { OCTAVERAISE }
| ".lowerOctave"  { OCTAVELOWER }
| "string" { STRING }
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "return" { RETURN }
| "int"     { INT }
| "bool"    { BOOL }
| "float"   { FLOAT }
| "string" { STRING }
```

```
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| '/' ((['A'-'G']['+''-''.']?|'R') as lxm) '/'  { TLIT(lxm) }
| '/' (digit | "-1" as lxm) '/'       { OLIT(int_of_string lxm) }
| '/' ((['s''e''q''h''w']['.']?) as lxm) '/'            { RLIT(lxm) }
| '"' ((ascii | esc)* as s)'"'                          { STRLIT(s) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )?   as lxm {
FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*       as lxm {
ID(lxm) }
(* make sure it can't be R or A B C D E F G *)
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
char)) }

and comment = parse
  ":)" { token lexbuf }
| _    { comment lexbuf }
```

8.4. parser.mly

```
/* Ocamlyacc parser for CFlat */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES DIVIDE
ASSIGN
%token LBRACK RBRACK
%token TONEACCESS OCTAVEACCESS RHYTHMACCESS
%token TONESET OCTAVESET RHYTHMSET
%token TONERAISE OCTAVERAISE OCTAVELOWER
%token MAKE
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE
%token INT BOOL FLOAT VOID NOTE STRING TONE OCTAVE RHYTHM
%token <int> LITERAL OLIT
%token <bool> BLIT
%token <string> ID FLIT STRLIT TLIT RLIT
%token EOF
```

```
%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT
%right TONERAISE OCTAVERAISE OCTAVELOWER
%right TONESET OCTAVESET RHYTHMSET
%right TONEACCESS OCTAVEACCESS RHYTHMACCESS

%%

program:
  decls EOF { $1 }

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

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
RBRACE
     { { typ = $1;
       fname = $2;
       formals = List.rev $4;
       locals = List.rev $7;
       body = List.rev $8 } }

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

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

typ:
```

```
    primitive_typ { PrimitiveType($1) }
  | array_typ     { $1 }

primitive_typ:
    INT     { Int   }
  | BOOL    { Bool  }
  | FLOAT   { Float }
  | VOID    { Void  }
  | NOTE    { Note  }
  | TONE    { Tone  }
  | OCTAVE  { Octave }
  | RHYTHM  { Rhythm }
  | STRING  { String }

array_typ:
  primitive_typ LBRACK RBRACK { ArrayType($1) }

literal:
    LITERAL          { Literal($1)           }
  | FLIT              { Fliteral($1)          }
  | BLIT             { BoolLit($1)           }
  | STRLIT           { StrLit($1)            }
  | notelit          { $1                    }
  | tlit             { $1                    }
  | olit             { $1                    }
  | rlit             { $1                    }

tlit:
    TLIT             { ToneLit($1)           }

olit:
    OLIT             { OctaveLit($1)         }

rlit:
    RLIT             { RhythmLit($1)         }

notelit:
    LPAREN tlit olit rlit RPAREN    { NoteLit($2, $3, $4)}
    /* allow for default values? */

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
```

```
   typ ID SEMI { ($1, $2) }

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

stmt:
    expr SEMI                                  { Expr $1               }
  | RETURN expr_opt SEMI                       { Return $2             }
  | LBRACE stmt_list RBRACE                    { Block(List.rev $2)    }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)        }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
                                             { For($3, $5, $7, $9)   }
  | WHILE LPAREN expr RPAREN stmt           { While($3, $5)         }

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

expr:
    literal          { $1 }
  | ID               { Id($1) }
  | ID ASSIGN expr   { Assign($1, $3)         }
  | ID LPAREN args_opt RPAREN { Call($1, $3)  }
  | ID TONEACCESS    { ToneAccess($1)         }
  | ID OCTAVEACCESS  { OctaveAccess($1)       }
  | ID RHYTHMACCESS  { RhythmAccess($1)       }
  | ID TONESET LPAREN expr RPAREN          { ToneSet($1, $4)
}
  | ID OCTAVESET LPAREN expr RPAREN        { OctaveSet($1, $4)
}
  | ID RHYTHMSET LPAREN expr RPAREN        { RhythmSet($1, $4)
}
  | ID TONERAISE LPAREN expr RPAREN        { ToneRaise($1, $4)
}
  | ID OCTAVERAISE LPAREN expr RPAREN      { OctaveRaise($1, $4)
}
  | ID OCTAVELOWER LPAREN expr RPAREN      { OctaveLower($1, $4)
}
  | LPAREN expr RPAREN { $2                   }
  | expr PLUS   expr { Binop($1, Add,   $3)   }
  | expr MINUS  expr { Binop($1, Sub,   $3)   }
  | expr TIMES  expr { Binop($1, Mult,  $3)   }
  | expr DIVIDE expr { Binop($1, Div,   $3)   }
```

```
  | expr EQ     expr { Binop($1, Equal, $3)   }
  | expr NEQ    expr { Binop($1, Neq,   $3)   }
  | expr LT     expr { Binop($1, Less,  $3)   }
  | expr LEQ    expr { Binop($1, Leq,   $3)   }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3)   }
  | expr AND    expr { Binop($1, And,   $3)   }
  | expr OR     expr { Binop($1, Or,    $3)   }
  | MINUS expr %prec NOT { Unop(Neg, $2)      }
  | NOT expr         { Unop(Not, $2)          }
  | MAKE LPAREN primitive_typ COMMA expr RPAREN   { MakeArray($3, $5)
}
  | ID LBRACK expr RBRACK ASSIGN expr             { ArrayAssign($1,
$3, $6) }
  | ID LBRACK expr RBRACK                 { ArrayAccess($1, $3) }

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

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

8.5. ast.ml

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

type primitive_typ = Int | Bool | Float | Void | Note | String | Tone
| Octave | Rhythm

type typ = PrimitiveType of primitive_typ | ArrayType of
primitive_typ

type bind = typ * string

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

type uop = Neg | Not

type expr =
    Literal of int
```

```
  | Fliteral of string
  | BoolLit of bool
  | StrLit of string
  | NoteLit of expr * expr * expr
  | ToneLit of string
  | OctaveLit of int
  | RhythmLit of string
  | ToneAccess of string
  | OctaveAccess of string
  | RhythmAccess of string
  | ToneSet of string * expr
  | OctaveSet of string * expr
  | RhythmSet of string * expr
  | ToneRaise of string * expr
  | OctaveRaise of string * expr
  | OctaveLower of string * expr

  | MakeArray of primitive_typ * expr
  | ArrayAssign of string * expr * expr
  | ArrayAccess of (string * expr)
  | Id of string
  | Assign of string * expr
  | Call of string * expr list
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Noexpr


type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

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

type program = bind list * func_decl list
```

```
(* Pretty-printing functions *)

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

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

let string_of_primitive_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Note -> "note"
  | Tone -> "tone"
  | Octave -> "octave"
  | Rhythm -> "rhythm"
  | String -> "string"

let rec string_of_typ = function
    PrimitiveType(t) -> string_of_primitive_typ t
  | ArrayType(t) -> (string_of_primitive_typ t) ^ "[]"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StrLit(l) -> l
  | NoteLit(t, o, r) -> string_of_expr t ^ ", " ^ string_of_expr o ^
", " ^ string_of_expr r
  | ToneLit(l) -> l
```

```
  | OctaveLit(l) -> string_of_int l
  | RhythmLit(l) -> l
  | ToneAccess(n) -> n ^ ".tone()"
  | OctaveAccess(n) -> n ^ ".octave()"
  | RhythmAccess(n) -> n ^ ".rhythm()"
  | ToneSet(n, e) -> n ^ ".tone(" ^ string_of_expr e ^ ")"
  | OctaveSet(n, e) -> n ^ ".octave(" ^ string_of_expr e ^ ")"
  | RhythmSet(n, e) -> n ^ ".rhythm(" ^ string_of_expr e ^ ")"
  | ToneRaise(n, e) -> n ^ ".raiseTone(" ^ string_of_expr e ^ ")"
  | OctaveRaise(n, e) -> n ^ ".raiseOctave(" ^ string_of_expr e ^ ")"
  | OctaveLower(n, e) -> n ^ ".lowerOctave(" ^ string_of_expr e ^ ")"

  | MakeArray(t, e) -> "make(" ^ string_of_primitive_typ t ^ "," ^
string_of_expr e ^ ")"
  | ArrayAssign(arr_name, e1, e2) -> arr_name ^ "[" ^ string_of_expr
e1 ^ "]" ^ "=" ^ string_of_expr e2
  | ArrayAccess(arr_name, e) -> arr_name ^ "[" ^ string_of_expr e ^
"]"
  | Id(s) -> s
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
"}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ;
" ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
string_of_stmt s
```

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

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

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.6. sast.ml

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

open Ast

type sexpr = typ * sx

and sx =
    SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SStrLit of string
  | SNoteLit of sexpr * sexpr * sexpr
  | SToneLit of string
  | SOctaveLit of int
  | SRhythmLit of string
  | SToneAccess of string
  | SOctaveAccess of string
  | SRhythmAccess of string
  | SToneSet of string * sexpr
  | SOctaveSet of string * sexpr
  | SRhythmSet of string * sexpr
  | SToneRaise of string * sexpr
  | SOctaveRaise of string * sexpr
  | SOctaveLower of string * sexpr

  | SId of string
```

```
  | SAssign of string * sexpr
  | SArrayAssign of string * sexpr * sexpr
  | SArrayAccess of string * sexpr
  | SMakeArray of primitive_typ * sexpr
  | SCall of string * sexpr list
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SFliteral(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SNoteLit(t, o, r) -> string_of_sexpr t ^ ", " ^ string_of_sexpr o
^ ", " ^ string_of_sexpr r
  | SToneLit(l) -> l
  | SOctaveLit(l) -> string_of_int l
  | SRhythmLit(l) -> l
  | SToneAccess(n) -> n ^ ".tone()"
  | SOctaveAccess(n) -> n ^ ".octave()"
  | SRhythmAccess(n) -> n ^ ".rhythm()"
  | SToneSet(n, e) -> n ^ ".tone(" ^ string_of_sexpr e ^ ")"
  | SOctaveSet(n, e) -> n ^ ".octave(" ^ string_of_sexpr e ^ ")"
```

```
  | SRhythmSet(n, e) -> n ^ ".rhythm(" ^ string_of_sexpr e ^ ")"
  | SToneRaise(n, e) -> n ^ ".raiseTone(" ^ string_of_sexpr e ^ ")"
  | SOctaveRaise(n, e) -> n ^ ".raiseOctave(" ^ string_of_sexpr e ^
")"
  | SOctaveLower(n, e) -> n ^ ".lowerOctave(" ^ string_of_sexpr e ^
")"

  | SMakeArray(t, e) -> "make(" ^ string_of_primitive_typ t ^ "," ^
string_of_sexpr e ^ ")"
  | SArrayAssign(n, e1, e2) -> n ^ "[" ^ string_of_sexpr e1 ^ "]" ^
"=" ^ string_of_sexpr e2
  | SArrayAccess(arr_name, e) -> arr_name ^ "[" ^ string_of_sexpr e ^
"]"
  | SStrLit(l) -> l
  | SId(s) -> s
  | SCall(f, el) ->
        f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el)
^ ")"
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SNoexpr -> ""
                      ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^
"}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n"
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ "
; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
```

```
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

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

8.7. semant.ml

```
(* Semantic checking for the Cflat compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

(* Array checking helpers *)
let match_array = function
    ArrayType(_) -> true
  | _ -> false

let check_array_or_throw typ a_name =
  if match_array typ then () else raise (Failure (a_name ^ " is not
an array"))

let get_array_type = function
    ArrayType(typ) -> typ
  | _ -> raise (Failure "invalid array type")

let check (globals, functions) =

  (* Verify a list of bindings has no void types or duplicate names
*)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
```

```
     (PrimitiveType(Void), b) -> raise (Failure ("illegal void " ^
kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      |    ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
       raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (**** Check global variables ****)

  check_binds "global" globals;

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

  (* Collect function declarations for built-in functions: no bodies
*)
  let built_in_decls =

    let add_bind map (name, tys) = StringMap.add name {
      typ = PrimitiveType(Void);
      fname = name;
      formals = tys;
      locals = []; body = [] } map
    in List.fold_left add_bind StringMap.empty
                              [ ("print", [(PrimitiveType(Int),
"x")]);
                                        ("printb",
[(PrimitiveType(Bool), "x")]);
                                        ("printf",
[(PrimitiveType(Float), "x")]);
                                        ("printbig",
[(PrimitiveType(Int), "x")]);
                                ("prints", [(PrimitiveType(String),
"x")]);
                                                   ("printn",
[(PrimitiveType(Note), "x")]);
                                ("printt", [(PrimitiveType(Tone),
"x")]);
                                ("printr", [(PrimitiveType(Rhythm),
"x")]);
                                ("printo", [(PrimitiveType(Octave),
"x")]);
```

```
                                ("playnote", [(PrimitiveType(Note),
"x"); (PrimitiveType(String), "y")]);
                                ("bplaynote", [(PrimitiveType(Note),
"x"); (PrimitiveType(Int), "y"); (PrimitiveType(String), "z")]);
                                ("playtrack", [(ArrayType(Note), "x");
(PrimitiveType(String), "y")]);
                                ]
  in

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

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

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

  let _ = find_func "main" in (* Ensure "main" is defined *)
  let check_function func =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "formal" func.formals;
    check_binds "local" func.locals;

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

```
    in

    let check_arr_assign lvaluet rvaluet err =
      if lvaluet = ArrayType(rvaluet) then rvaluet else raise
(Failure err)
   in

    (* Build local symbol table of variables for this function *)
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add
name ty m)
                  StringMap.empty (globals @ func.formals @
func.locals )
    in

    (* Return a variable from our local symbol table *)
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^
s))
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec expr = function
        Literal  l -> (PrimitiveType(Int), SLiteral l)
      | Fliteral l -> (PrimitiveType(Float), SFliteral l)
      | BoolLit l -> (PrimitiveType(Bool), SBoolLit l)
      | NoteLit (t, o, r) -> (PrimitiveType(Note), SNoteLit (expr t,
expr o, expr r))
      | ToneLit l -> (PrimitiveType(Tone), SToneLit l)
      | OctaveLit l -> (PrimitiveType(Octave), SOctaveLit l)
      | RhythmLit l -> (PrimitiveType(Rhythm), SRhythmLit l)
      | ToneAccess n -> (PrimitiveType(Tone), SToneAccess n)
      | OctaveAccess n -> (PrimitiveType(Octave), SOctaveAccess n)
      | RhythmAccess n -> (PrimitiveType(Rhythm), SRhythmAccess n)
      | ToneSet(n, e) -> (PrimitiveType(Tone), SToneSet (n, expr e))
      | OctaveSet(n, e) -> (PrimitiveType(Octave), SOctaveSet (n,
expr e))
      | RhythmSet(n, e) -> (PrimitiveType(Rhythm), SRhythmSet (n,
expr e))
      | ToneRaise(n, e) -> (PrimitiveType(Note), SToneRaise (n, expr
e))
      | OctaveRaise(n, e) -> (PrimitiveType(Note), SOctaveRaise (n,
expr e))
      | OctaveLower(n, e) -> (PrimitiveType(Note), SOctaveLower (n,
expr e))
```

```
      | StrLit l -> (PrimitiveType(String), SStrLit l)
      | Noexpr -> (PrimitiveType(Void), SNoexpr)
      | Id s -> (type_of_identifier s, SId s)
      | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = "
^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
      | Unop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
            Neg when t = PrimitiveType(Int) || t =
PrimitiveType(Float) -> t
          | Not when t = PrimitiveType(Bool) -> PrimitiveType(Bool)
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
          let (t1, e1') = expr e1
          and (t2, e2') = expr e2 in
          (* All binary operators require operands of the same type
*)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand
types *)
          let ty = match op with
            Add | Sub | Mult | Div when same && t1 =
PrimitiveType(Int)   -> PrimitiveType(Int)
          | Add | Sub | Mult | Div when same && t1 =
PrimitiveType(Float) -> PrimitiveType(Float)
          | Equal | Neq            when same               ->
PrimitiveType(Bool)
          | Less | Leq | Greater | Geq
                     when same && (t1 = PrimitiveType(Int) || t1 =
PrimitiveType(Float)) -> PrimitiveType(Bool)
          | And | Or when same && t1 = PrimitiveType(Bool) ->
PrimitiveType(Bool)
          | _ -> raise (Failure ("illegal binary operator " ^
                         string_of_typ t1 ^ " " ^ string_of_op op ^
" " ^
```

```
                          string_of_typ t2 ^ " in " ^ string_of_expr
e))
            in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | MakeArray(t, e) as ex ->
          let (t', e') = expr e in
          if t' = PrimitiveType(Int)
            then (ArrayType(t), SMakeArray(t, (t',e')))
          else raise (Failure ("illegal make, must provide integer
size for " ^  string_of_expr e))
      | ArrayAccess (a_name, e) ->
          let t = type_of_identifier a_name
          and (t', e') = expr e
          in ignore (check_array_or_throw t a_name);
          (PrimitiveType(get_array_type t), SArrayAccess(a_name, (t',
e')))
      | ArrayAssign (a_name, e1, e2) as ex ->
          let lt = (type_of_identifier a_name)
          and (t', e1') = expr e1
          and (rt, e2') = expr e2 in
          (PrimitiveType(get_array_type lt), SArrayAssign(a_name,
(t', e1'), (rt, e2')))
          (* let lt = type_of_identifier a_name
          and (t', e1') = expr e1
          and (rt, e2') = expr e2 in
            let err = "illegal assignment " ^ string_of_typ lt ^ " =
" ^
              string_of_typ rt ^ " in " ^ string_of_expr ex
            in (check_arr_assign lt rt err, SArrayAssign(a_name, (t',
e1'), (rt, e2'))) *)
      | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length
^
                            " arguments in " ^ string_of_expr call))
          else let check_call (ft, _) e =
            let (et, e') = expr e in
            let err = "illegal argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^
string_of_expr e
            in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.formals args
          in (fd.typ, SCall(fname, args'))
```

```
    in

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

    (* Return a semantically-checked statement i.e. containing sexprs
*)
    let rec check_stmt = function
        Expr e -> SExpr (expr e)
      | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
check_stmt b2)
      | For(e1, e2, e3, st) ->
        SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
      | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
      | Return e -> let (t, e') = expr e in
        if t = func.typ then SReturn (t, e')
        else raise (
          Failure ("return gives " ^ string_of_typ t ^ " expected " ^
            string_of_typ func.typ ^ " in " ^ string_of_expr e))

         (* A block is correct if each statement is correct and
nothing
            follows any Return statement.  Nested blocks are
flattened. *)
      | Block sl ->
          let rec check_stmt_list = function
              [Return _ as s] -> [check_stmt s]
            | Return _ :: _   -> raise (Failure "nothing may follow a
return")
            | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten
blocks *)
            | s :: ss         -> check_stmt s :: check_stmt_list ss
            | []              -> []
          in SBlock(check_stmt_list sl)

    in (* body of check_function *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      slocals  = func.locals;
```

```
      sbody = match check_stmt (Block func.body) with
                      SBlock(sl) -> sl
                  | _ -> raise (Failure ("internal error: block didn't
become a block?"))
    }
  in (globals, List.map check_function functions)
```

8.8. codegen.ml

```
(*

Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in

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

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
```

```
  and void_t     = L.void_type   context in
  let str_t      = L.pointer_type i8_t
  and i32_ptr_t  = L.pointer_type i32_t
  and i8_ptr_t  = L.pointer_type i8_t in
  let named_struct_note_t = L.named_struct_type context
"named_struct_note_t" in
  ignore (L.struct_set_body named_struct_note_t [| L.pointer_type
i8_t; L.i32_type context; L.pointer_type i8_t |] false);

  (* Return the LLVM type for a CFlat type *)
  let ltype_of_primitive_typ = function
      A.PrimitiveType(A.Int)   -> i32_t
    | A.PrimitiveType(A.Bool)  -> i1_t
    | A.PrimitiveType(A.Float) -> float_t
    | A.PrimitiveType(A.Void)  -> void_t
    | A.PrimitiveType(A.Note)  -> named_struct_note_t
    | A.PrimitiveType(A.Tone)  -> str_t
    | A.PrimitiveType(A.Octave)  -> i32_t
    | A.PrimitiveType(A.Rhythm)  -> str_t
    | A.PrimitiveType(A.String) -> str_t
  in

  let ltype_of_typ = function
      A.PrimitiveType(t) ->
ltype_of_primitive_typ(A.PrimitiveType(t))
    | A.ArrayType(t) -> L.pointer_type (ltype_of_primitive_typ
(A.PrimitiveType(t)))
  in

  (* Create a map of global variables after creating each *)
  let global_vars : L.llvalue StringMap.t =
    let global_var m (t, n) =
      let init = match t with
          A.PrimitiveType(A.Float) -> L.const_float
(ltype_of_primitive_typ t) 0.0
        | _ -> L.const_int (ltype_of_primitive_typ t) 0
      in StringMap.add n (L.define_global n init the_module) m in
    List.fold_left global_var StringMap.empty globals in

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
      L.declare_function "printf" printf_t the_module in

  let printbig_t : L.lltype =
```

```
      L.function_type i32_t [| i32_t |] in
  let printbig_func : L.llvalue =
      L.declare_function "printbig" printbig_t the_module in

  let play_note_t : L.lltype =
      L.function_type i32_t [| L.pointer_type named_struct_note_t ;
L.pointer_type i8_t |] in
  let play_note_func : L.llvalue =
      L.declare_function "play_note" play_note_t the_module in

  let bplay_note_t : L.lltype =
      L.function_type i32_t [| L.pointer_type named_struct_note_t ;
i32_t ; L.pointer_type i8_t |] in
  let bplay_note_func : L.llvalue =
      L.declare_function "bplay_note" bplay_note_t the_module in

  let play_note_arr_t : L.lltype =
      L.function_type i32_t [| L.pointer_type named_struct_note_t ;
L.pointer_type i8_t |] in
  let play_note_array_func : L.llvalue =
      L.declare_function "play_note_arr" play_note_arr_t the_module
in

  let change_tone_t : L.lltype =
      L.function_type i32_t [| L.pointer_type named_struct_note_t ;
i32_t ; i32_t |] in
  let change_tone_r : L.lltype =
      L.return_type change_tone_t in
  let change_tone_func : L.llvalue =
      L.declare_function "change_tone" change_tone_t the_module in

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

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

    let int_format_str = L.build_global_stringptr "%d\n" "fmt"
builder
    and note_format_str = L.build_global_stringptr "/%s/ /%d/ /%s/\n"
"fmt" builder
    and float_format_str = L.build_global_stringptr "%g\n" "fmt"
builder
    and tone_format_str = L.build_global_stringptr "/%s/\n" "fmt"
builder
    and octave_format_str = L.build_global_stringptr "/%d/\n" "fmt"
builder
    and rhythm_format_str = L.build_global_stringptr "/%s/\n" "fmt"
builder
    and str_format_str = L.build_global_stringptr "%s\n" "fmt"
builder in

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

      (* Allocate space for any locally declared variables and add
the
       * resulting registers to our map *)
      and add_local m (t, n) =
           let local_var = L.build_alloca (ltype_of_typ t) n builder
           in StringMap.add n local_var m
      in

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

```
      List.fold_left add_local formals fdecl.slocals
    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    (* ========================= Array Constructors
========================= *)
    (* TAKEN/ADAPTED FROM C?
http://www.cs.columbia.edu/~sedwards/classes/2017/4115-fall/reports/I
nception.pdf
      Whenever an array is made, we malloc an additional 16 bytes of
metadata,
      which contains size and length information. This allows us to
implement
      len() in a static context, and opens several possibilities
including
      array concatenation, dynamic array resizing, etc.
      The layout will be:
      +--------------+--------------+----------+--------+
      | element size | size (bytes) | len[int] | elem1  | ...
      +--------------+--------------+----------+--------+
    *)

    let elem_size_offset = L.const_int i32_t (-3)
    and size_offset = L.const_int i32_t (-2)
    and len_offset = L.const_int i32_t (-1)
    and metadata_sz = L.const_int i32_t 12 in  (* 12 bytes overhead
*)

    let put_meta body_ptr offset llval builder =
      let ptr = L.build_bitcast body_ptr i32_ptr_t "i32_ptr_t"
builder in
      let meta_ptr = L.build_gep ptr [| offset |] "meta_ptr" builder
in
      L.build_store llval meta_ptr builder
    in

    let get_meta body_ptr offset builder =
      let ptr = L.build_bitcast body_ptr i32_ptr_t "i32_ptr_t"
builder in
```

```
      let meta_ptr = L.build_gep ptr [| offset |] "meta_ptr" builder
in
      L.build_load meta_ptr "meta_data" builder
    in

    let meta_to_body meta_ptr builder =
      let ptr = L.build_bitcast meta_ptr i8_ptr_t "meta_ptr" builder
in
      L.build_gep ptr [| (L.const_int i8_t (12)) |] "body_ptr"
builder
    in

    let body_to_meta body_ptr builder =
      let ptr = L.build_bitcast body_ptr i8_ptr_t "body_ptr" builder
in
      L.build_gep ptr [| (L.const_int i8_t (-12)) |] "meta_ptr"
builder
    in

    (* make array *)
    let make_array element_t len builder =
      let element_sz = L.build_bitcast (L.size_of element_t) i32_t
"b" builder in
      let body_sz = L.build_mul element_sz len "body_sz" builder in
      let malloc_sz = L.build_add body_sz metadata_sz "make_array_sz"
builder in
      let meta_ptr = L.build_array_malloc i8_t malloc_sz "make_array"
builder in
      let body_ptr = meta_to_body meta_ptr builder in
      ignore (put_meta body_ptr elem_size_offset element_sz builder);
      ignore (put_meta body_ptr size_offset malloc_sz builder);
      ignore (put_meta body_ptr len_offset len builder);
      L.build_bitcast body_ptr (L.pointer_type element_t)
"make_array_ptr" builder
    in


    (*
=============================================================================
========= *)

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) = match e with
           SLiteral i  -> L.const_int i32_t i
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
```

```
      | SFliteral l -> L.const_float_of_string float_t l
      | SNoteLit (t, o, r) -> let t' = expr builder t and
                                o' = expr builder o and
                                r' = expr builder r in
                                L.const_named_struct
named_struct_note_t [| t'; o'; r' |]
      | SToneLit t ->  L.build_global_stringptr (t ^ "\x00")
"tone_ptr" builder
      | SOctaveLit o ->  L.const_int i32_t o
      | SRhythmLit r ->  L.build_global_stringptr (r ^ "\x00")
"rhythm_ptr" builder
      | SStrLit l   -> L.build_global_stringptr (l ^ "\x00")
"str_ptr" builder
      | SNoexpr     -> L.const_int i32_t 0
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign (s, e) -> let e' = expr builder e in
                            ignore(L.build_store e' (lookup s)
builder); e'
      | SBinop ((A.PrimitiveType(Float), _) as e1, op, e2) ->
                           let e1' = expr builder e1 and e2' = expr
builder e2 in
                            ( match op with
                              A.Add     -> L.build_fadd
                            | A.Sub     -> L.build_fsub
                            | A.Mult    -> L.build_fmul
                            | A.Div     -> L.build_fdiv
                            | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
                               | A.Neq     -> L.build_fcmp
L.Fcmp.One
                               | A.Less    -> L.build_fcmp
L.Fcmp.Olt
                               | A.Leq     -> L.build_fcmp
L.Fcmp.Ole
                               | A.Greater -> L.build_fcmp
L.Fcmp.Ogt
                               | A.Geq     -> L.build_fcmp
L.Fcmp.Oge
                               | A.And | A.Or ->
                                 raise (Failure "internal error:
semant should have rejected and/or on float")
                            ) e1' e2' "tmp" builder
      | SBinop (e1, op, e2) ->
                           let e1' = expr builder e1 and e2' = expr
builder e2 in
                            ( match op with
```

```
                              A.Add     -> L.build_add
                            | A.Sub     -> L.build_sub
                            | A.Mult    -> L.build_mul
                            | A.Div     -> L.build_sdiv
                            | A.And     -> L.build_and
                            | A.Or      -> L.build_or
                            | A.Equal   -> L.build_icmp L.Icmp.Eq
                            | A.Neq     -> L.build_icmp L.Icmp.Ne
                            | A.Less    -> L.build_icmp L.Icmp.Slt
                            | A.Leq     -> L.build_icmp L.Icmp.Sle
                            | A.Greater -> L.build_icmp L.Icmp.Sgt
                            | A.Geq     -> L.build_icmp L.Icmp.Sge
                            ) e1' e2' "tmp" builder
      | SUnop (op, ((t, _) as e)) ->
                          let e' = expr builder e in
                            ( match op with
                              A.Neg when t = A.PrimitiveType(A.Float)
-> L.build_fneg
                            | A.Neg                  -> L.build_neg
                            | A.Not                  -> L.build_not
                            ) e' "tmp" builder
      | SToneAccess n -> let tb = L.build_struct_gep (lookup n) 0
"@tone" builder in
                            L.build_load tb ".tone" builder
      | SOctaveAccess n -> let ob = L.build_struct_gep (lookup n) 1
"@octave" builder in
                            L.build_load ob ".octave" builder
      | SRhythmAccess n -> let rb = L.build_struct_gep (lookup n) 2
"@rhythm" builder in
                            L.build_load rb ".rhythm" builder
      | SToneSet (n, e) -> let tb = L.build_struct_gep (lookup n) 0
"@tone" builder in
                            let e' = expr builder e in
                            ignore(L.build_store e' tb builder); e'
      | SOctaveSet (n, e) -> let ob = L.build_struct_gep (lookup n) 1
"@octave" builder in
                            let e' = expr builder e in
                            ignore(L.build_store e' ob builder); e'
      | SRhythmSet (n, e) -> let rb = L.build_struct_gep (lookup n) 2
"@rhythm" builder in
                            let e' = expr builder e in
                            ignore(L.build_store e' rb builder); e'

      | SOctaveRaise (n, e) -> let ob = L.build_struct_gep (lookup n)
1 "@octave" builder in
```

```
                              let e1' = expr builder e and e2' =
L.build_load ob ".octave" builder in
                              let sum = L.build_add e1' e2'
"add_octaves" builder in
                              let x = L.build_store sum ob builder in
                              L.build_load (lookup n)
"raise_octave_of_this_note" builder

      | SOctaveLower (n, e) -> let ob = L.build_struct_gep (lookup n)
1 "@octave" builder in
                              let e1' = expr builder e and e2' =
L.build_load ob ".octave" builder in
                              let sum = L.build_sub e2' e1'
"add_octaves" builder in
                              let x = L.build_store sum ob builder in
                              L.build_load (lookup n)
"lower_octave_of_this_note" builder
```

```
      | SToneRaise (n, e) -> let e' = expr builder e in
                              let b' = L.const_int i32_t 0 in
                              ignore(L.build_call change_tone_func [|
(lookup n) ; e' ; b' |]
                              "change_tone" builder);
                              L.build_load (lookup n)
"raise_tone_of_this_note" builder
                              (* let r' = L.build_alloca change_tone_r
"returned_by_change_tone" builder in *)

                              (* let n' = L.build_call change_tone_func
[| (lookup n) ; e' ; b' |]
                              "change_tone" builder in
                              L.build_load n' "returned_tone_ptr"
builder *)
```

```
                            (* let nv = L.build_load n'
"returned_tone_ptr" builder in
                            L.build_load nv "returned_tonelit"
builder *)



                            (*let ns = L.build_load nv
"returned_tonelit" builder in
                            expr builder (A.PrimitiveType(A.Tone),
SToneLit ns) *)



                            (* let nv = L.build_load n'
"from_build_call" builder in *)
                            (* let rv = L.build_load r'
"returned_value" builder in *)


                            (* ignore(L.build_store n' (lookup n)
builder); n' :) *)



                            (* ignore(L.build_store rv (lookup n)
builder); rv *)
                            (* ignore(L.build_store nv (lookup n)
builder); nv *)

                            (* expr builder (PrimitiveType(Note),
SAssign (n, (PrimitiveType(Note), n'))) *)

                            (* let t' = L.build_struct_gep n' 0
"@tone" builder and
                                o' = L.build_struct_gep n' 1
"@octave" builder and
                                r' = L.build_struct_gep n' 2
"@rhythm" builder in
                            let nl = L.const_named_struct
named_struct_note_t [| t'; o'; r' |] in
                            ignore(L.build_store nl (lookup n)
builder); nl *)
```

```
                            (* let n' = L.build_call change_tone_func
                              [| (lookup n) ; e' ; b' |]
                              "change_tone" builder in
                            ignore(L.build_store n' (lookup n)
builder); n' *)

                           (* L.build_global_stringptr ("hiiii" ^
"\x00") "tone_ptr" builder  *)
                           (* L.build_extractvalue (lookup n) 0
".tone" builder *)
                            (* in
                            L.value_name tv *)
                            (* L.build_global_stringptr tv "tone_ptr"
builder *)
                           (* let nv = lookup n in
                            let tv = L.build_extractvalue nv 0
".tone" builder in
                            let tvv = L.const_extractvalue tv [| 0 |]
in
                            L.build_load tvv "note.tone" builder *)
                           (* L.const_extractvalue (lookup n) [| 0 |]
*)
                           (* let nv = L.build_load (lookup n) n
builder in
                            L.const_extractvalue nv [| 0 |] *)
                            (* L.build_extractvalue nv 0 ".tone"
builder *)
                            (* let tb = L.build_struct_gep nv 0
"@tone" builder in
                            L.build_load tb ".tone" builder *)

      | SMakeArray (t, e) -> let len = expr builder e
                            in make_array (ltype_of_primitive_typ
(A.PrimitiveType(t))) (len) builder
      | SArrayAssign (arr_name, idx_expr, val_expr) ->
                              let idx = (expr builder idx_expr)
                              and assign_val = (expr builder
val_expr) in
                              let llname = arr_name ^ "[" ^
L.string_of_llvalue idx ^ "]" in
                              let arr_ptr = lookup arr_name in
                              let arr_ptr_load = L.build_load arr_ptr
arr_name builder in
                              let arr_gep = L.build_in_bounds_gep
arr_ptr_load [|idx|] llname builder in
```

```
                              L.build_store assign_val arr_gep
builder
      | SArrayAccess (arr_name, idx_expr) ->
        let idx = expr builder idx_expr in
        let llname = arr_name ^ "[" ^ L.string_of_llvalue idx ^ "]"
in
        let arr_ptr_load =
            let arr_ptr = lookup arr_name in
            L.build_load arr_ptr arr_name builder in
        let arr_gep = L.build_in_bounds_gep arr_ptr_load [|idx|]
llname builder in
          L.build_load arr_gep (llname ^ "_load") builder
      | SCall ("print", [e]) | SCall ("printb", [e]) ->
           L.build_call printf_func [| int_format_str ; (expr
builder e) |]
           "printf" builder
      | SCall ("printbig", [e]) ->
           L.build_call printbig_func [| (expr builder e) |]
"printbig" builder
      | SCall ("printf", [e]) ->
           L.build_call printf_func [| float_format_str ; (expr
builder e) |]
           "printf" builder
      | SCall ("prints", [e]) ->
           L.build_call printf_func [| str_format_str ; (expr
builder e) |]
           "printf" builder
      | SCall ("printn", [e]) -> let (_, SId n) = e in
                              let t' = expr builder
(A.PrimitiveType(A.Tone), SToneAccess n)
                              and o' = expr builder
(A.PrimitiveType(A.Octave), SOctaveAccess n)
                              and r' = expr builder
(A.PrimitiveType(A.Rhythm), SRhythmAccess n) in
        L.build_call printf_func [| note_format_str ; t'; o'; r' |]
        "printf" builder
      | SCall ("printt", [e]) ->
           L.build_call printf_func [| tone_format_str ; (expr
builder e) |]
           "printf" builder
      | SCall ("printo", [e]) ->
        L.build_call printf_func [| octave_format_str ; (expr builder
e) |]
        "printf" builder
      | SCall ("printr", [e]) ->
```

```
        L.build_call printf_func [| rhythm_format_str ; (expr builder
e) |]
        "printf" builder
      | SCall ("playnote", [e1 ; e2]) -> let (_, SId n) = e1 in
        L.build_call play_note_func [| (lookup n) ; (expr builder e2)
|] "play_note" builder
      | SCall ("bplaynote", [e1 ; e2 ; e3]) -> let (_, SId n) = e1 in
        L.build_call bplay_note_func [| (lookup n) ; (expr builder
e2) ; (expr builder e3)|] "bplay_note" builder
      | SCall ("playtrack", [e1 ; e2]) -> let (_, SId n) = e1 in
        L.build_call play_note_array_func [| (expr builder e1) ;
(expr builder e2) |] "play_note_array" builder
      | SCall (f, args) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
           let llargs = List.rev (List.map (expr builder) (List.rev
args)) in
           let result = (match fdecl.styp with
                        A.PrimitiveType(A.Void) -> ""
                      | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder
    in

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

    (* Build the code for the given statement; return the builder for
       the statement's successor (i.e., the next instruction will be
built
       after the one generated by this call) *)

    let rec stmt builder = function
     SBlock sl -> List.fold_left stmt builder sl
      | SExpr e -> ignore(expr builder e); builder
      | SReturn e -> ignore(match fdecl.styp with
                            (* Special "return nothing" instr *)
```

```
                              A.PrimitiveType(A.Void) ->
L.build_ret_void builder
                              (* Build return statement *)
                            | _ -> L.build_ret (expr builder e)
builder );
                      builder
      | SIf (predicate, then_stmt, else_stmt) ->
                              let bool_val = expr builder predicate
in
                              let merge_bb = L.append_block context
"merge" the_function in
                              let build_br_merge = L.build_br
merge_bb in (* partial function *)

                              let then_bb = L.append_block context
"then" the_function in
                                add_terminal (stmt (L.builder_at_end
context then_bb) then_stmt)
                                build_br_merge;

                              let else_bb = L.append_block context
"else" the_function in
                                add_terminal (stmt (L.builder_at_end
context else_bb) else_stmt)
                                build_br_merge;

                                ignore(L.build_cond_br bool_val
then_bb else_bb builder);
                                L.builder_at_end context merge_bb

      | SWhile (predicate, body) ->
                              let pred_bb = L.append_block context
"while" the_function in
                                ignore(L.build_br pred_bb builder);

                              let body_bb = L.append_block context
"while_body" the_function in
                                add_terminal (stmt (L.builder_at_end
context body_bb) body)
                                (L.build_br pred_bb);

                              let pred_builder = L.builder_at_end
context pred_bb in
                              let bool_val = expr pred_builder
predicate in
```

```
                                let merge_bb = L.append_block context
"merge" the_function in
                                  ignore(L.build_cond_br bool_val
body_bb merge_bb pred_builder);
                                  L.builder_at_end context merge_bb
      (* Implement for loops as while loops *)
      | SFor (e1, e2, e3, body) -> stmt builder
                                  ( SBlock [SExpr e1 ; SWhile (e2,
SBlock [body ; SExpr e3]) ] )

                                  in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

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

  List.iter build_function_body functions;
  the_module
```

8.9. cflatapi

8.9.1. cflatapi.h

```
#ifndef CFLATAPI_H_
#define CFLATAPI_H_

/* Note struct */
struct note{
    char* tlit;
    int olit;
    char* rlit;
}note;
/*Allocates space for a note struct*/
struct note *new_note(char *tone, int octave, char *rhythm);

/*
** CFLAT USER FUNCTION DECLARATIONS
```

```
*/

/* Plays a single note
**INPUT: Takes in a pointer to a single note struct
**OUTPUT: A midifile called "(i/b)hellonote.mid" that plays the note
*/
void play_note(struct note *note_ptr, char *filename);
void bplay_note(struct note *n, int beat, char *filename); /*
bplay_note takes in beat (beats/min) */
void iplay_note(struct note *n, int instrument);  /* iplay_note takes
in an instrument 1-127*/


/* Plays a single note
**INPUT: Takes in a pointer to an array of note struct pointers
**OUTPUT: A midifile called "notearray.mid" that plays a C Major
scale.
*/
void play_note_arr(struct note *note_arr, char *filename);
void bplay_note_arr(struct note *note_arr[], int beat); /*
bplay_note_arr takes in beat (beats/min) */
void iplay_note_arr(struct note *note_arr[], int instrument);
void ibplay_note_arr(struct note *note_arr[], int instrument, int
beat);

void play_tracks(int i, ...);

#endif
```

8.9.2. cflatapi.c

```
/*The CFlat API. Codegen should call functions from this library */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#ifndef  __APPLE__
#include <malloc.h>
#endif
#include "midifile.h"
#include "cflatapi.h"

/*Mallocs space for and initializes a new note struct */
struct note *new_note(char *tone, int octave, char *rhythm){
```

```
    struct note *n = malloc(sizeof(struct note));
    if (n == NULL) return NULL;

    strcpy(n->tlit, tone);
    n->olit = octave;
    strcpy(n->rlit, rhythm);

    return n;
}


/* INTERNAL FUNCTIONS, CFLAT USERS SHOULD NOT CALL THESE */

/*INPUT: Takes in a pointer to a single note struct and pointer to a
midi_file */
/*OUTPUT: No output but it will add the note to the midi_file */
void add_note(struct note *note_ptr, MIDI_FILE *mf, int track);

/*INPUT: Takes in a pointer to an array of note struct, pointer to a
midi_file */
/*OUTPUT: No output but it will add track with notes to the midifile
*/
void add_track(struct note *note_arr[], MIDI_FILE *mf, int track);


void play_note(struct note *note_ptr, char *filename) {
    MIDI_FILE *mf;
    char name[100];
    strcpy(name, filename);
    char midi[] = ".mid";
    strcat(name, midi);
     if ((mf = midiFileCreate(name, TRUE))){
        midiTrackAddProgramChange(mf, 1,
MIDI_PATCH_ACOUSTIC_GRAND_PIANO);
          add_note(note_ptr, mf, 1);
          midiFileClose(mf);
     }
}

void bplay_note(struct note *note_ptr, int beat, char *filename) {
MIDI_FILE *mf;
     if ((mf = midiFileCreate("output.mid", TRUE))){
        midiSongAddTempo(mf, 1, beat);
```

```
        midiTrackAddProgramChange(mf, 1,
MIDI_PATCH_ACOUSTIC_GRAND_PIANO);
           add_note(note_ptr, mf, 1);
           midiFileClose(mf);
           }
}

void iplay_note(struct note *note_ptr, int instrument) {
MIDI_FILE *mf;
     if ((mf = midiFileCreate("hellonoteinst.mid", TRUE))){

        midiTrackAddProgramChange(mf, 1, instrument);
           add_note(note_ptr, mf, 1);
           midiFileClose(mf);
           }
}

void play_note_arr(struct note *note_arr, char *filename){

    MIDI_FILE *mf;
    char name[100];
    strcpy(name, filename);
    char midi[] = ".mid";
    strcat(name, midi);
    if ((mf = midiFileCreate(name, TRUE))){
        int i = 0;
        for (i = 0; i < 23; i++) {
/*
            printf("%s%d\n", "OCTAVE,", (*note_arr) -> olit);
            printf("%s%s\n", "RHYTHM,", (*note_arr) -> rlit);
            printf("%s%s\n", "OCTAVE,", (*note_arr) -> tlit); */

            add_note((&note_arr[i]), mf, 1);
        }

        midiFileClose(mf);
    }
}

void bplay_note_arr(struct note *note_arr[], int beat){
    MIDI_FILE *mf;
    if ((mf = midiFileCreate("helloarraybeat.mid", TRUE))){
        midiSongAddTempo(mf, 1, beat);
        while (*note_arr){
            add_note((*note_arr), mf, 1);
```

```
            note_arr++;
        }
        midiFileClose(mf);
    }
}

void iplay_note_arr(struct note *note_arr[], int instrument){
    MIDI_FILE *mf;
    if ((mf = midiFileCreate("helloarrayinst.mid", TRUE))){
        midiTrackAddProgramChange(mf, 1, instrument);
        while (*note_arr){
            add_note((*note_arr), mf, 1);
            note_arr++;
        }
        midiFileClose(mf);
    }
}

void add_note(struct note *note_ptr, MIDI_FILE *mf, int track){

    char *tlit = note_ptr -> tlit;
    int olit = note_ptr -> olit;
    char *rlit = note_ptr -> rlit;

    int miditone = 0;
    int is_rest = 0;
    char tone = tlit[0];
    printf("tone: %s\n", tlit);
    printf("octave: %d\n", olit);
    printf("rhythm: %s\n", rlit);
    if (tone == 'R') {is_rest = 1;}

    else if (tone == 'C') {miditone = 0;}
    else if (tone == 'D') {miditone = 2;}
    else if (tone == 'E') {miditone = 4;}
    else if (tone == 'F') {miditone = 5;}
    else if (tone == 'G') {miditone = 7;}
    else if (tone == 'A') {miditone = 9;}
    else if (tone == 'B') {miditone = 11;}
    else {printf("%s", "This is not a valid tone.");}

    int accidental = 0;

    /*char acc = tlit[1];*/
    char acc = tlit[1];
```

```
    if (acc == '-') {accidental = -1;}
    else if (acc == '+') {accidental = 1;}
    else if (acc == '.') {accidental = 0;}
    /*  else {printf("%s\n", "This is not an allowable accidental
value.");}*/

    miditone = (miditone + accidental)%12;    /*Accounts for any
wraparound needed for B# or Cflat*/


    /*CONVERTING CFLAT RHYTHM => MIDI RHYTHM*/
    int midirhythm = MIDI_NOTE_CROCHET;
    int dotted = 0;
    if (strlen(rlit) > 1 && rlit[1] == '.') {dotted = 1;}   /*checks
for dotted value*/

    char rhythm = rlit[0];
    if (rhythm == 's') {midirhythm = MIDI_NOTE_SEMIQUAVER;}
    else if (rhythm == 'e') {midirhythm = MIDI_NOTE_QUAVER;}
    else if (rhythm == 'q') {midirhythm = MIDI_NOTE_CROCHET;}
    else if (rhythm == 'h') {midirhythm = MIDI_NOTE_MINIM;}
    else if (rhythm == 'w') {midirhythm = MIDI_NOTE_BREVE;}

    else {printf("%s%c\n", "Rhythm is not a valid rhythm.", rhythm);}


    if (dotted) { midirhythm += midirhythm/2; }  /* adds the dotted
portion */

    /*CONVERTING CFLAT olit => MIDI OCTAVE*/
    int midioctave = olit;
    if (olit >= 0 && olit <= 10){
        midioctave *= 12;
    }
    /*DEFAULT TIME SIGNATURE */
    midiSongAddSimpleTimeSig(mf, track, 4, MIDI_NOTE_CROCHET);
    midiFileSetTracksDefaultChannel(mf, track, MIDI_CHANNEL_1);
    midiTrackAddText(mf, track, textLyric, tlit);

    int volume = MIDI_VOL_MEZZO;
    if (is_rest){
        volume = 0;
    }
```

```
    midiTrackAddNote(mf, track, miditone + midioctave, midirhythm,
volume, TRUE, FALSE);

}

void add_track(struct note *note_arr[], MIDI_FILE *mf, int track){

    printf("%d", track);
    while (*note_arr){
        add_note((*note_arr), mf, track);
        note_arr++;
    }
}

void play_tracks(int num_tracks, ...){
    MIDI_FILE *mf;
    va_list valist;
    va_start(valist, num_tracks);
    struct note **track;
    int i;

    if ((mf = midiFileCreate("playTracks.mid", TRUE))){

        for (i = 0; i < num_tracks; i++){
            track = va_arg(valist, struct note **);
            add_track(track, mf, i);
        }

        midiFileClose(mf);
    }
}

/* char **change_tone(char *tlit, int incr, int is_lower){

    int miditone = 0;
    int is_rest = 0;
    char tone = tlit[0];

    if (tone == 'R') {is_rest = 1;}
    else if (tone == 'C') {miditone = 0;}
    else if (tone == 'D') {miditone = 2;}
    else if (tone == 'E') {miditone = 4;}
    else if (tone == 'F') {miditone = 5;}
    else if (tone == 'G') {miditone = 7;}
    else if (tone == 'A') {miditone = 9;}
```

```
    else if (tone == 'B') {miditone = 11;}
    else {printf("%s", "This is not a valid tone.");}

    int accidental = 0;
    char acc = tlit[1];

    if (acc == '-') {accidental = -1;}
    else if (acc == '+') {accidental = 1;}
    else if (acc == '.') {accidental = 0;}
    else {printf("%s\n", "This is not an allowable accidental
value.");}

    if(is_lower){miditone = miditone + accidental - incr;}
    else{miditone = miditone + accidental + incr;}

    miditone = ((miditone)%12+12)%12;

    char *toneMap[] = {"C", "C+", "D", "D+", "E", "F", "F+", "G",
"G+", "A", "A+", "B"};

    return &toneMap[miditone];
} */

void change_tone(struct note *note, int incr, int is_lower){

    char *tlit = note -> tlit;
    int olit = note -> olit;

    /* char *newtlit;
    int newolit;
    char *samerlit = note -> rlit;
    struct note* newnoteptr;
    struct note newnote; */

    int miditone = 0;
    int is_rest = 0;
    char tone = tlit[0];

    if (tone == 'R') {is_rest = 1;}
    else if (tone == 'C') {miditone = 0;}
    else if (tone == 'D') {miditone = 2;}
    else if (tone == 'E') {miditone = 4;}
    else if (tone == 'F') {miditone = 5;}
    else if (tone == 'G') {miditone = 7;}
    else if (tone == 'A') {miditone = 9;}
```

```
    else if (tone == 'B') {miditone = 11;}
    else {printf("%s", "This is not a valid tone.");}

    int accidental = 0;
    char acc = tlit[1];

    if (acc == '-') {accidental = -1;}
    else if (acc == '+') {accidental = 1;}
    else if (acc == '.') {accidental = 0;}
    else {printf("%s\n", "This is not an allowable accidental
value.");}

    if(is_lower){miditone = miditone + accidental - incr;}
    else{miditone = miditone + accidental + incr;}

    if (is_lower){
        while(miditone < 0){
            miditone += 12;
            olit -=1;
        }
    }else{
        while (miditone > 11){
            miditone -= 12;
            olit +=1;
        }
    }

    if (is_rest) {tlit = tlit;}

    char *toneMap[] = {"C", "C+", "D", "D+", "E", "F", "F+", "G",
"G+", "A", "A+", "B"};

    /* strcmp(newtlit, toneMap[miditone]);
    newolit = olit;
    newnoteptr = new_note(newtlit, newolit, samerlit);
    newnote = *newnoteptr; */

    strcpy(tlit, toneMap[miditone]);

    note -> tlit = tlit;
    note -> olit = olit;

}
```

8.10. Makefile

```
CC=gcc
LINK = $(CC)
CFLAGS = -O2 -ansi -Wall
LDFLAGS = -s

# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
     ./testall.sh

# "make all" builds the executable as well as the "printbig" library
designed
# to test linking external code

.PHONY : all
all : cflat.native printbig.o cflatapi.o midifile.o

# "make cflat.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by
including
# packages, enabling warnings
#
# See
https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

cflat.native :
     opam config exec -- \
     ocamlbuild -use-ocamlfind cflat.native

# "make clean" removes all generated files

.PHONY : clean
clean :
     ocamlbuild -clean
     rm -rf testall.log ocamlllvm
     rm -f *.o *.out *.mid *.ll *.s *. *.exe *.diff *.err

# Testing the "printbig" example

printbig: printbig.c
```

```
	cc -o printbig -DBUILD_TEST printbig.c

# midifiles
midifile.o: midifile.c	midifile.h

cflatapi.o: cflatapi.c cflatapi.h

# Building the tarball
TESTS = arr_assign_int arr_assign_note arr_concat_int arr_concat_int
arr_decl_int arr_make_int arr_make_note\
	  binop0 binop1 bplaynote for0 for1 helloworld0 helloworld1
if0 if1 if2 int microcfloat1 note noteatt1_declare\
	  noteatt2_get noteatt3_getset playnote printfloat
tone_features unop0 while0

FAILS = assign_int_note
# FAILS = microcprint

MIDI = playnote bplaynote

TESTFILES = $(TESTS:%=test-%.cf) $(TESTS:%=test-%.out) \
	    $(FAILS:%=fail-%.cf) $(FAILS:%=fail-%.err) \
		$(MIDI:%=midi-%.cf) $(MIDI:%=midi-%.mid)

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags cflat.ml
parser.mly \
	README scanner.mll semant.ml testall.sh \
	printbig.c cflatapi.c midifile.c arcade-font.pbm font2c \
	Dockerfile \
	$(TESTFILES:%=tests/%)

cflat.tar.gz : $(TARFILES)
	cd .. && tar czf cflat/cflat.tar.gz \
	  $(TARFILES:%=cflat/%)
```

8.11. testall.sh

```
#!/bin/sh

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

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the CFlat compiler.  Usually "./cflat.native"
# Try "_build/cflat.native" if ocamlbuild was unable to create a
symbolic link.
CFLAT="./cflat.native"
#CFLAT="_build/cflat.native"

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to
difffile
Compare() {
```

```
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
     SignalError "$1 differs"
     echo "FAILED $1 differs from $2" 1>&2
    }
}

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

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
     SignalError "failed: $* did not report an error"
     return 1
    }
    return 0
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
```

```
    Run "$CFLAT" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
"${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o"
"cflatapi.o" "midifile.o" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

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

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

    echo -n "$basename..."

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

     generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
    Run "$CFLAT" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
"${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "cflatapi.o"
"midifile.o" &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare output.mid tests/${basename}.mid ${basename}.diff
```

```
     # Report the status and clean up the generated files

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

```
    fi
}

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

shift `expr $OPTIND - 1`

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

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

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

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/midi-*.cf tests/test-*.cf tests/fail-*.cf"
fi

for file in $files
do
    case $file in
    *midi-*)
        CheckMidi $file 2>> $globallog
        ;;
```

```
      *test-*)
          Check $file 2>> $globallog
          ;;
      *fail-*)
          CheckFail $file 2>> $globallog
          ;;

      *)
          echo "unknown file type $file"
          globalerror=1
          ;;
    esac
done

exit $globalerror
```

8.12. tests

8.12.1. fail-assign_int_note.cf

```
int main()
{
      int i;
      note n;
      i = n;
      return 0;
}
```

8.12.2. midi-bplaynote.cf

```
int main()
{
      note n;
      string filename;
      n = ( /C-/ /4/ /s/ );
      filename = "output";
      bplaynote(n, 60, filename);

      return 0;
}
```

8.12.3. midi-bplaynote1.cf

```
int main()
{
      note n;
      string filename;
      n = ( /C-/ /5/ /w/ );
      filename = "output";
      bplaynote(n, 360, filename);

      return 0;
}
```

8.12.4. midi-play_arr.cf

```
int main()
{
  note n;
  note m;
  note[] note_arr;

  n = ( /C-/ /4/ /q/ );
  m = ( /E/ /5/ /w/ );
  note_arr = make(note, 3);

  note_arr[0] = n;
  note_arr[1] = n;
  note_arr[2] = m;

  playtrack(note_arr, "playarr");
  return 0;
}
```

8.12.5. midi-playnote.cf

```
int main()
{
      note n;
      string file;
      n = ( /C-/ /4/ /s./ );
      file = "output";
      playnote(n, file);

      return 0;
}
```

8.12.6. midi-playnote1.mid

```
int main()
{
      note n;
      string filename;
      n = ( /A+/ /7/ /h./ );
      filename = "output";
      playnote(n, filename);

      return 0;
}
```

8.12.7. test-arr_decl_int.cf

```
int main()
{
  int[] int_arr;
  return 0;
}
```

8.12.8. test-arr_decl_note.cf

```
int main()
{
  note[] note_arr;
  return 0;
}
```

8.12.9. test-arr_make_int.cf

```
int main()
{
  int[] intarr;
  intarr = make(int, 3);
  intarr[0] = 1;
  intarr[1] = 2;
  intarr[2] = 3;
  (: print(int_arr);           should print out the array literal :)
  print(intarr[1]);
```

```
  return 0;
}
```

8.12.10. test-arr_make_note.cf

```
int main()
{
  note n;
  note m;
  note k;
  note[] notearr;

  n = ( /C-/ /4/ /q/ );
  m = ( /E/ /5/ /w/ );
  notearr = make(note, 6);

  notearr[0] = n;
  notearr[1] = m;
  k = notearr[1];
  (: notearr[2] = m; :)

  printn(k);
  return 0;
}
```

8.12.11. test-binop0.cf

```
int main() {
     print(1 + 2);
     print(1 - 2);
     print(1 * 2);
     print(100 / 2);
     print(99);
     return 0;
}
```

8.12.12. test-binop1.cf

```
int main() {
    printb(1 == 2);
    printb(1 == 1);
    printb(1 != 2);
```

```
    printb(1 != 1);
    printb(1 < 2);
    printb(2 < 1);
    printb(1 <= 2);
    printb(1 <= 1);
    printb(2 <= 1);
    printb(1 > 2);
    printb(2 > 1);
    printb(1 >= 2);
    printb(1 >= 1);
    printb(2 >= 1);
    return 0;
}
```

8.12.13. test-crazy_compbool.cf

```
int main()
{
    note n;
    note m;

    n = (/C/ /4/ /s./);
    m = (/F/ /5/ /q/);

    printb( (n.tone() != m.tone()) && (n.tone() != m.tone()) );
    printb( (n.tone() != m.tone()) || (n.tone() == m.tone()) );

     return 0;
}
```

8.12.14. test-for0.cf

```
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}
```

8.12.15. test-for1.cf

```
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
```

8.12.16. test-helloworld0.cf

```
int main() {
     prints("Hello World");
     return 0;
}
```

8.12.17. test-helloworld1.cf

```
int main() {
     string s;
     s = "I am alive";
     prints(s);
     return 0;
}
```

8.12.18. test-if0.cf

```
int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
```

8.12.19. test-if1.cf

```
int main()
```

```
{
  if (false) {
      print(42);
  }
  print(17);
  return 0;
}
```

8.12.20. test-int.cf

```
int main()
{
  int i;
  i = 3;
  return 0;
}
```

8.12.21. test-microcfloat1.cf

```
int main()
{
      float a;
      a = 3.14159267;
      printf(a);
      return 0;
}
```

8.12.22. test-note.cf

```
int main()
{
      note n;
      n = (/C-/ /4/ /s./);
      printn(n);

      return 0;
}
```

8.12.23. test-noteatt1_declare.cf

```
int main()
```

```
{
	tone t;
	octave o;
	rhythm r;

	t = /C-/;
	o = /4/;
	r = /s./;

	printt(t);
	printo(o);
	printr(r);

	return 0;
}
```

8.12.24. test-noteatt2_get.cf

```
int main()
{
	note n;
	tone t;
	octave o;
	rhythm r;

	n = ( /C-/ /0/ /s./ );

	t = n.tone();
	printt(t);

	o = n.octave();
	printo(o);

	r = n.rhythm();
	printr(r);

	return 0;
}
```

8.12.25. test-noteatt3_getset.cf

```
int main()
{
```

```
    note n;
    note v;
    tone t;
    octave o;
    rhythm r;

    n = ( /C-/ /4/ /s./ );

    t = n.tone();
    o = n.octave();
    r = n.rhythm();

    v.tone(t);
    printt(v.tone());

    v.tone(/A/);
    printt(v.tone());

    v.octave(o);
    printo(v.octave());

    v.octave(/5/);
    printo(v.octave());

    v.rhythm(r);
    printr(v.rhythm());

    v.rhythm(/h/);
    printr(v.rhythm());

    return 0;
}
```

8.12.26. test-octave_compbool.cf

```
int main()
{
    note n;
    note m;

    octave o;
    octave v;

    n = (/C/ /4/ /s./);
```

```
    m = (/F/ /5/ /q/);

    o = /3/;
    v = /3/;

    printb( n.octave() == m.octave() );
    printb( n.octave() != m.octave() );

    printb( o == v );
    printb( o != v );

     return 0;
}
```

8.12.27. test-printfloat.cf

```
int main() {
     float f;
     f = 3.14;
     printf(f);
     return 0;
}
```

8.12.28. test-raiselowerOctave.cf

```
int main()
{
  note n;
  n = (/C/ /4/ /s./);
  printo(n.octave());

  n = n.raiseOctave(1);
  printo(n.octave());

  n = n.lowerOctave(3);
  printo(n.octave());

  return 0;
}
```

8.12.29. test-rhythm_compbool.cf

```
int main()
{
    note n;
    note m;

    rhythm t;
    rhythm v;

    n = (/C/ /4/ /s./);
    m = (/F/ /5/ /q/);

    t = /h/;
    v = /h/;

    printb( n.rhythm() == m.rhythm() );
    printb( n.rhythm() != m.rhythm() );

    printb( t == v );
    printb( t != v );

     return 0;
}
```

8.12.30. test-unop0.cf

```
int main() {
     printb(true);
     printb(false);
     printb(true && true);
     printb(true && false);
     printb(false && true);
     printb(false && false);
     printb(true || true);
     printb(true || false);
     printb(false || true);
     printb(false || false);
     printb(!false);
     printb(!true);
     print(-10);
     print(--42);
     return 0;
}
```

8.12.31. test-while0.cf

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```