Improv Language 🎵

Alice Zhang (ayz2105)
Emily Li (el2895)
Josh Choi (jc4881)
Natalia Dorogi (ngd2111)

Columbia University School of Engineering & Applied Sciences
Programming Languages and Translators
Professor Edwards

April 26, 2021

Table of Contents

1. Introduction

The Improv language is a language inspired by our admiration of music and appreciation for algorithms. It was initially designed for improvising over music files, and in the process we realized how useful it can be to hear algorithms - something that is missing from the current tool box. We call this: music humanities representation of algorithms. In other words, our language allows for programmers and consumers of the MIDI files our programs create to understand how code sounds - this is especially helpful for algorithms. *picture Columbia data structures students listening to each sorting algorithm.* This is further interesting to explore the relationship between music and math. A lot of mathematicians are musicians but it is more difficult to understand how math *sounds,* which is enabled through our language Improv.

Improvisation, made simple, is founded upon arranging notes from the pentatonic scale of a song's key over its instrumental. We are building the Improv language to synthesize the music file of an improvised solo based on the user's inputs and specifications. The idea is to generate a musical melody that would sound good layered on top of a backing track instrumental, like a solo improvisation.

When using Improv, the user may choose to set a key, e.g. C major, and a tempo in BPM (beats per minute), e.g. 100 BPM. The declared key dictates the notes that the user has access to, namely those from the key's pentatonic scale, e.g. C major includes C, D, E, G, and A notes. The use of the pentatonic scale guarantees that the progression of notes harmonizes well over any instrument with the corresponding key and BPM. Other specifications the user can set include varying note lengths, e.g. eighth, quarter, half notes, and different rhythm patterns, e.g. repeated note lengths.

1.1 The Music Basics

In music, notes have a tone and a duration. Tone is characterized by the pitch. There are 12 different possible tones including C, F#, and Ab, and many possible durations including quarter notes, half notes, and eighth notes. Combined, the tone and duration of notes creates music of all types and forms.

A note in Improv is defined by a tone and a rhythm. The tone ranges from 0 to 4, each integer representing a note in the pentatonic scale. The rhythm is a two character string that represents different note durations such as "ei" and "hf". We are able to construct pentatonic-style improvisation by mapping integer values to pentatonic scale notes,

thus creating a musical melody that would sound like its improvising over a backing track of the same key.

The key and BPM must be inputted to generate the MIDI file. The key determines which scale is used when generating the pentatonic scale, and includes values like CMAJ, DMIN, and F#MIN. The key describes both the pitch and the mode, major or minor, which are necessary inputs to selecting the right pentatonic scale. The BPM (beats-per-minute) determines how fast the tempo is and how long each duration ratio is. An eighth note in 60 BPM will be much shorter than an eighth note in 200 BPM.

2. Language Tutorial

Welcome to Improv! Improv uses an intuitive syntax that is similar to C.
Let's get started. 🎵

2.1 Setup

Improv was developed with Ocaml. In the following section, we will walk through how to create your environment to develop the Improv code. Docker is very helpful for this.

Install Homebrew
run in terminal

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install Docker
run in terminal

```
brew install docker
```

Install LLVM
run in terminal

```
opam init
opam install llvm.3.8
eval 'opam config env'
```

Clone the Improv repository

In /src directory, run `make all` to create the machine executable `improv.native`

2.2 Run Test Cases

- In the main improv folder, run `./testAll.sh`

2.3 Hello World

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

In the above code snippet,

- `func int main() {}` is the function declaration. Each .impv file has a main function, as this is where program execution begins. Improv functions have defined return types, e.g. main returns a value of integer type. If the function took parameters, they would be listed within the parentheses. The curly braces define the scope of the function's body.
- `prints()` is a built-in Improv function that prints string expressions, in this case "Hello World". Improv uses different print functions to print different types, such as `prints()` for integers, `printn()` for notes, `printa()` for integer arrays, and `printNoteArr()` for integer arrays.
- Because we defined the type of the return value to be an integer, we `printa()` `0` at the end of the function.
- The print and return statements each end with a semicolon, indicating the end of the statement.

2.4 Basic Arithmetic and Variables

```
func int main() {
     int a;
     int b;
     int c;

     a = 3;
     b = 8;

     printi(a+1);
```

```
      printi(b-4);
      printi(a*b);
      printi(b/2);
      printi(b%a);
}
```

In the above code snippet,

- Local variables are defined at the beginning of the function body and can be used anywhere in the scope of the function. Improv is strongly and statically typed, so the variables are initialized with their names and corresponding types. Above, `int a; int b; int c;` are all local variables and can be used throughout the body of `main()`.
- Variable assignment is done with an equal sign and can only be executed if the types of the expressions on the left and right sides are equivalent. For example, `a` is a variable of type integer, and `3` is an integer literal.
- printi is a built-in Improv function that prints the value of integer expressions. Above, `printi(a+1)` prints 4 because `a` is equal to 3.
- Improv supports many basic arithmetic operations, including addition, subtraction, multiplication, division, modulus, etc. These operations can be used if both operands are of type integer.

2.5 Conditionals and Loops

```
func int main() {
      int i;
      int j;
      int k;

      i = 5;
      if (i%2 == 1) {
            prints(“odd”);
      } else {
            prints(“even”);
      }

      for (j=0; j<10; j=j+1) {
            printi(j);
      }
```

```
      k = 5;
      while (k > 0) {
            printi(k);
            k = k - 1;
      }

      return 0;
}
```

In the above code snippet,

- Improv includes if-else conditionals with the `if` and `else` keywords and a boolean conditional expression. The first block is executed if the `if` conditional evaluates to `true`, otherwise the `else` block is executed. In the case above, because `i` equals 5, `i%2==1` evaluates to `true`, and “odd” is printed.
- Improv also includes for loops with the `for` keyword and three expressions passed in. The first expression gives the initial value of the iterator variable, the second gives the boolean conditional to indicate the end of iteration, and the third expression iterates the iterator variable. Above, the iterator variable `j` is first set to equal 0, then is passed through one iteration of the for loop block. That is, 0 is printed out. Then, `j` is iterated based on the third expression—incremented by 1, and checked against the second conditional expression. `j` passes through because 1 < 10, and 1 is printed out. This repeats until the iteration where `j` is set equal to 10, and 10 < 10 evaluates to false. The for loop then ends, and 10 is not printed out. Integers 0 through 9 are printed out by the example for loop.
- Improv supports while loops with the `while` keyword and a boolean conditional expression. The `while` block is repeatedly executed until the conditional evaluates to false. In the case above, `k` is set to 5, and 5 > 0, so 5 is printed out and `k` is decremented by 1. This repeats until `k` is decremented to equal 0 and does not pass the conditional check of 0 > 0. The `while` block then ends, with the integers 5, 4, 3, 2, 1 having been printed.

2.6 Arrays

```
func int main() {
      note[] note_arr;
      int[] int_arr;
```

```
      int[] int_arr2;
      int[] int_arr3;
      int i;

      note_arr = [<3, "wh">, <4, "qr">];
      for(i=0; i<2; i=i+1) {
            printn(note_arr[i]);
      }

      int_arr = [1, 3, 5];
      int_arr[0] = 10;
      printa(int_arr);

      int_arr2 = [2, 4];
      int_arr3 = append(int_arr, int_arr2);

      return int_arr3[3];
}
```

In the above code snippet,

- Arrays are the central data structures in Improv. They are initialized at the top of the function block along with the local variables. Arrays can be defined in any primitive type—`string, int, bool, note`. Array elements are accessed with square brackets and the index of the element, and array elements are reassigned to new values with square brackets and an index along with the equal sign for assignment. Above, the note array is iterated over and printed, while the first element of the integer array is reassigned a new value.
- Improv has a built-in `append` function that takes in two arrays of the same type as parameters and appends them. For example, `int_arr3` is `int_arr` appended to `int_arr2`, so it will be set equal to `[1, 3, 5, 2, 4]`. As a result, the function returns 2.

2.7 Functions

```
func string foo(int x) {
      printi(x);
```

```
      return "done";
}

func int main() {
      int a;

      foo(5);

      a = 3;
      foo(a);

      return 0;
}
```

In the above code snippet,

- Function calls in Improv are done with the function name followed by parentheses. The function can take in parameters of typed variables, and a value is passed in for the parameter when the function is called. For example, `foo(int x)` takes in as a parameter an integer named `x`. When `foo` is called in `main()`, 5 is passed in as the parameter for the first call to `foo`, then 3 is passed in for the second call to `foo`.

2.8 Music

```
func int main() {
  note[] arr;
  arr = [<1, "qr">, <1, "qr">, <1, "qr">, <4, "qr">,
    <0, "qr">, <0, "qr">, <4, "hf">, <3, "qr">,
    <3, "qr">, <2, "qr">, <2, "qr">, <1, "hf">];
  render(arr, "mcd.mid", 13, 120);
  return 0;
}
```

In the above snippet,

- Improv allows users to render note arrays into MIDI files using the built-in render function. The note array is passed in first, then the output file name, followed by the key—represented by an integer—and the tempo in beats per minute (BPM)—also represented by an integer. In the case above, the `render` function

would output a file called “mcd.mid” that plays the note array defined by arr in C minor key—represented by 13—and with BPM of 120.

3. Language Manual

3.1 Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators.

In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator-pairs.

If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

3.2 Comments

The syntax of comments draw from Java. A single-line comment is marked by `//`, and a multiline-comment by `/* */`. Nested multi-line comments are not supported, but single-line comments may be nested in a single-line or multi-line comment.

```
// This is a single-line  comment.
int x = 5;
/* This is a
multi -line  comment.
//  Nested  single -line  comment.
*/
```

3.3 Identifiers

Identifiers may be any combinations of letters, numbers, and underscores for function and variables. For example,

```
my_improv
fun_thing_2
99 _bottles
```

Identifiers may not start with an int, use a dash (`-`), nor start or end with an underscore (`_`).

3.4 Keywords

The keywords that are reserved, and may not be used otherwise. These include:
Data types — `note, int, bool, string, none`
Boolean logic — `and, or, not, true, false`
Program structure — `main, func, in, if, else, for, while, return`
Structures, operations — `array, append, render, print`

3.5 Syntax

High level view of Improv syntax:

Entrypoint	main method	`func int main() {}`

Expressions	literals
	variables
	assignment
	function call
	if else
	for
	while
		`c`
		`x`
		`=`
		`foo()`
		`if {} else {}`
		`for {}`
		`while {}`

Primitive Types	note
	integer
	boolean
	string
	none
		`note n;`
		`int i;`
		`bool b;`
		`string s;`
		`func none foo() {}`

Structures	array	`[]`

3.6 Operators

The following subsections outline rules pertaining to operators in Improv. Arithmetic operators `(+, -, *, /, %)` are left-to-right associative with `*` and `/` having highest precedence.

3.6.1 Integer Operators

`+`	Addition	`int + int -> int`
`-`	Subtraction	`int + int -> int`
`*`	Multiplication	`int * int -> int`

`/`	Division	`int / int -> int`
`%`	Modulus	`int / int -> int`
`>`	Greater than	`int > int -> bool`
`>=`	Greater than or equal	`int >= int -> bool`
`<`	Less than	`int < int -> bool`
`<=`	Less than or equal	`int <= int -> bool`
`==`	Equality	`int == int -> bool`
`!=`	Not equality	`int != int -> bool`

3.6.2 Boolean Operators

`and`	Conjunction	`bool and bool -> bool`
`or`	Disjunction	`bool and bool -> bool`
`not`	Negation	`not bool -> bool`

3.7 Type System

3.7.1 Basic data types

Boolean: `bool` is a 8-bit boolean variable that may be true or false

Integers: `int` is literal 32-bit signed

String: `string` sequence of ASCII characters, enclosed by `" "`, such as `"hello world!"`

3.7.2 Music data types

Note: `note` is a struct data type encompassing two data fields: a `tone` and a `rhythm`.

Tone: `tone` is the pitch of the note. It is represented as a 32-bit integer that can either be an int literal or expression. It must contain values between 0-5, with a 0 representing a rest note and 1-5 corresponding to the 5 notes on the pentatonic scale.

Rhythm: `rhythm` is the length of the note. It is represented as a string; i.e. `wh` = whole note, `hf` = half note, `qn` = quarter note, `ei` = eighth note, `sx` = sixth note

3.7.3 Structures

Array: `array` represents an array of the same type and is mutable. These are literals enclosed in square brackets `[]`. Empty arrays must specify a type. Arrays may be accessed using indexing: `array[index]`.

Array examples include:

```
note[] a_riff;
string[] some_strings;
int[] empty_arr;

this_is_a_riff = [<3, "qr">, <2, "qr">, <1, "qr">, <3, "qr">, <4,
"qr">, <3, "qr">, <2, "qr">, <3, "qr">, <1, "hf">];
printn(a_riff[3]) //  prints <1, qr>;

some_strings = ["this", "is", "an", "arr", "of", "string"];
```

3.8 Control Flow

`if else`

Reserved keywords `if` and `else` are used for conditional statements and selection, and execute if the conditioned boolean expression evaluates to true. Expression body is enclosed by curly brackets such that:

```
if (expr){
      statement;
} else {
      statement;
}
```

`for`

Iteration is conducted using reserved keyword `for` along with three expressions, similarly to Java for loops:

```
for (expr; bool_expr; expr){
      statement;
```

```
}
```

while

Iteration is conducted using the reserved keyword `while`, which creates a loop for whenever a given boolean expression is true:

```
while (bool_expr) {
      statement;
}
```

3.9 Standard Library Functions

render

Renders a MIDI file of the music created that users can play

```
render(note[] arr, string filename, int key, int tempo);
```

printmidi

Prints a textual representation of a given MIDI file

```
printmidi("test.mid");
```

printi

Prints the enclosed integer expression.

```
int i;
i = 10;
printi(5);
printi(5+3);
printi(i+20);
```

prints

Prints the enclosed string expression.

```
string s;
s = "Hey Prof";
prints("Hello World");
printi(s);
```

printn

Prints the enclosed note expression.

```
note n;
n = <1, "qr">;
printn(<3, "wh">);
printn(n);
```

printa

Prints the enclosed int array expression.

```
int[] int_arr;
int_arr = [1, 2, 3, 4, 5];
printa(int_arr);
```

printNoteArr

Prints the enclosed Note array expression.

```
note[] note_arr;
note_arr = [<3, "qr">, <4, "wh">, <2, "wh">];
printNoteArr(note_arr);
```

append

Appends two arrays and returns the appended array.

```
note[] a1;
note[] a2;
note[] a3;
a1 = [<3, "qr">, <4, "wh">, <2, "wh">];
a2 = [<1, "hf">, <2, "sx">, <5, "ei">];
a3 = append(a1, a2);
```

4. Project Plan

4.1 Process Planning

We first met shortly after the first week of class to talk about what different ideas we were interested in for our language. That cadence then turned into a twice a week meeting to discuss our ideas, planning, and progress. In order to track our planning from week to week we used a shared Notion page that included a timeline and pointers that we picked up from class and office hours. In our timeline on Notion page we set deadlines and broke up tasks into small parts - this was helpful so that the work was digestible (and we were able to move tasks to complete more often, which is good for the ego). Our Teaching Assistant Hans was able to guide us in tasks that we should be

working on, which gave us a good understanding of staying on track throughout the semester. In addition to small tasks, we tracked the class set deadlines including the Proposal, Language Reference Manual / Parser, and Hello World. For our Proposal, we submitted a second version after there was a change in our team and we subsequently pivoted in our topic.

4.2 Meetings

We met as a team twice a week; one time as a group and one time as a group with our Teaching Assistant Hans on Tuesday nights. The structure of meetings ranged week to week as we generally worked in sprints in groups of two on different aspects of our compiler. As aforementioned, our Notion page was central to tracking meetings, and we used a Google Drive folder to keep track of meeting notes with our Teaching Assistant Hans. Meetings with Hans consisted of talking through our progress and problems we were encountering, as well as checking that we were in the right direction and on a good cadence for the semester. We made notes of all his advice and worked hard to bring progress to each of our check ins.

Meetings of just our team evolved during the semester from being on research (understanding music for our non-musically inclined members), understanding what did and did not exist (and what holes we could fill with our language), what kind of features to include, actually coding, taking a step back and thinking if we really should include all those features, and then iterating through the latter steps until we created a successful demo. In the process, we received a lot of mentorship and assistance from Hans, Harry and Evan who all were incredibly helpful in debugging and problem solving, as well as learning more in the process.

Outside of meetings, our team communicated via Facebook Messenger. This was a good place for us to drop relevant links that were helpful for the group, as well as ask quick questions - since two heads are better than one. This was especially helpful to share the design decisions that were made as we progressed through pair programming.

4.3 Development

During the development of our language, we made “macro” decisions at the beginning - ones that were more at a high level of what we thought would make the most sense including structures to include, types, and operators to name a few. We all found lectures very helpful to understand MicroC’s files and how each worked within the big

picture, as well as just reading through the code of MicroC. At the beginning of the project, we spent more time on zoom together going through code as it was important to understand, and easier to do all together. This was mostly due to our inexperience as a team with functional languages and OCaml - though as time progressed it became much more natural. As mentioned, we used a pair programming process with that group leading the development and owning a file.

4.4 Testing

As reiterated in class time and time again, we took an approach to test often after we implemented each new functionality to our language. This was imperative because it made it so we could isolate what was not working and focus our energy on that. After all we got to know how the errors would be very not descriptive so this was a productive way to mitigate the uncertainty around what worked and what did not.

4.5 Style Guide

As a team, we were guided by the following style while working on code.

4.5.1 Git requirements

- Pulling most recent must be done before pushing so that no changes were lost
- No two people would work on the same file from two different computers at the same time - zoom and safe in person meetings were utilized for pair programming
- All code must compile before pushing new changes
- Commit messages must be included and be understandable for the rest of the team

4.5.2 OCaml

We did not follow as strict of a style guide for our development in OCaml; for example sometimes the ‘in’ in ‘let in’ would be on the same or a separate line.

- We used comments often to denote what was happening in lines of code, especially code that was potentially unclear
- Fixed warnings as development progressed so that we could find the important errors fast
- Pattern matching blocks always occurred on a new line
- We used descriptive names for functions and variables; nothing was random

4.5.3 C

- Opening curly braces when defining functions do not begin on a new line
- Tabs are 2 spaces
- Variable names and function names are camel case
- Two enters between functions
- The else starts on the same line as the ending curly bracket of the corresponding if statement

4.6 Project Timeline

01/31/2021	First meeting to discuss ideas
2/03/2021	Submitted project proposal
02/14/2021	Team & project change </3
02/18/2021	Submitted updated project proposal
02/22/2021	Created Github repository
02/22/2021	Finished Scanner
03/19/2021	Finished Parser and AST
04/20/2021	Hello world
04/24/2021	MIDI file linked
04/25/2021	Finished Codegen
04/25/2021	Wrote demo files
04/25/2021	Project presentation
04/26/2021	Final report

4.7 Team Roles and Responsibilities

A lot of the roles became fluid as the project progressed, and each teammate worked on multiple aspects of the compiler. In the beginning of the project there was a lot more entire group coding - which later became narrowed to pair coding (using one computer to work when possible to safely meet).

Natalia Dorogi - Project Manager

Scanner, Parser, Semantic Check, presentation

Emily Li - System Architect

Scanner, Parser, Semantic Check, Codegen, C libraries

Josh Choi - Language Guru

Scanner, AST

Alice Zhang - Tester

Parser, AST, Semantic Check, Codegen

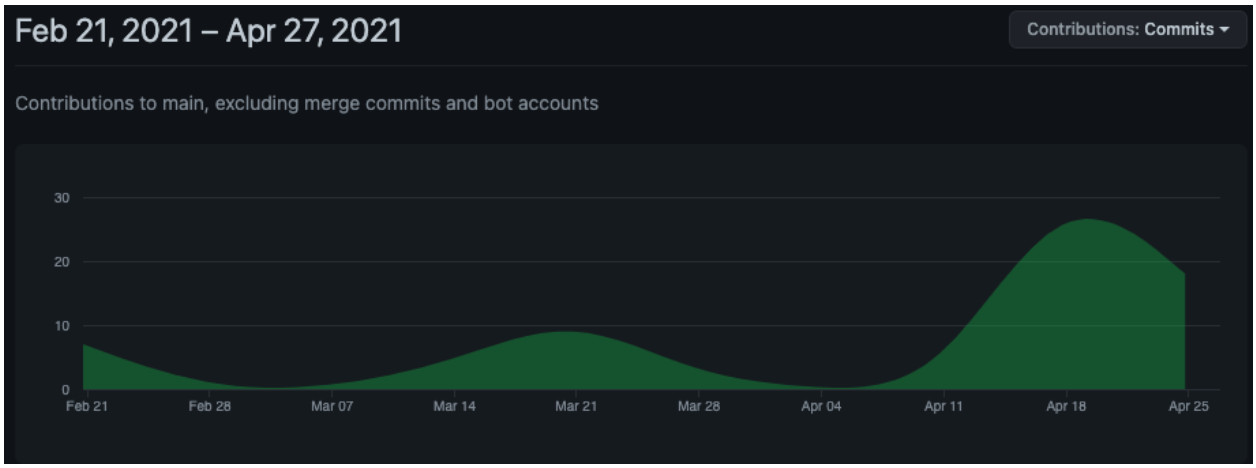
4.8 Software Development Tools

The following tools were utilized in order to successfully complete Improv:

- Project Management: Notion, Google Drive
- Languages: OCaml, LLVM, C
- Parser: Ocamlyacc, Ocamllex
- Compiling and Testing: Ocamlbuild
- Programming Editor: Virtual Studio Code
- Version Control: Github
- Documentation and Presentation: Google Docs, Overleaf, Google Slides
- Communication: Messenger

4.9 Project Log

Below is a summary of commits throughout the project and a more nuanced code log can be found in the Appendix.



5. Architectural Design

5.1 Block Diagram of Translator

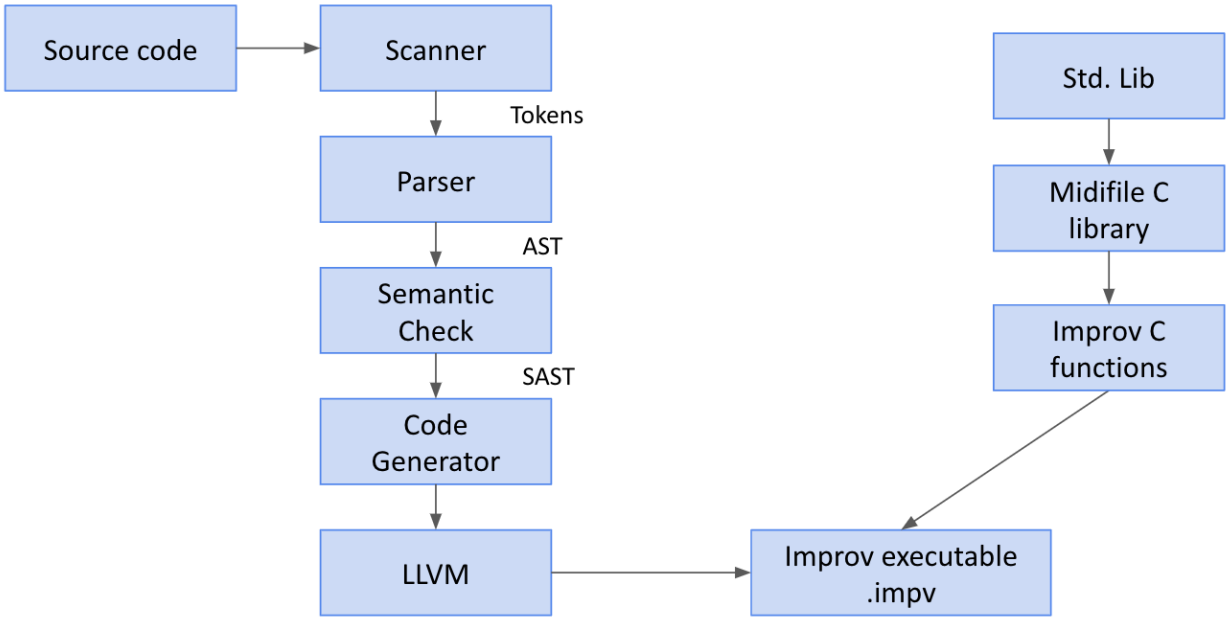


Figure 1: Architecture of Improv compiler

5.2 Scanner `scanner.mll`

The scanner is written in OCamlLex and takes in the Improv source code and generates tokens to identify keywords, operators, literals and other language conventions. It ignores line and block comments and any whitespace in the code. The tokens created by the scanner are then used in the Parser.

5.3 Parser and AST `parser.mly, ast.ml`

The parser is written in OCamlYacc and evaluates tokens generated by the scanner and creates an Abstract Syntax Tree (AST) by specifying precedence and matching recognized tokens. The grammar is defined in the parser using productions and rules. The source code is successfully parsed if it is syntactically correct.

5.4 Semantic Checking `semant.ml`

The semantic check is written in OCaml and checks for proper typing from the AST. If all the types are correct, then it creates a semantically-checked AST (SAST).

5.5 Code Generation `codegen.ml`

The code generator builds the LLVM intermediate representation instructions for the language using the semantically-checked AST.

5.6 C Libraries `midifile.c, midifile.h, midiinfo.h, improv.c, improv.h`

The improv C functions are built on top of the MidiFile C library. improv.h contains the struct data types such as notes and arrays, as well as a predefined pentatonic scale for each key. improv.c consists of built-in functions and helper functions to perform rendering a MIDI file, printing notes and arrays, appending arrays and other operations. It is linked with the LLVM bytecode in the binary executable file.

5.7 Implementation Roles

1. Scanner - Emily, Josh, Natalia
2. Parser - Natalia, Emily, Alice
3. AST - Alice, Josh
4. Semantic Check - Natalia, Emily, Alice
5. Codegen - Alice, Emily
6. C libraries - Emily

6. Test Plan

6.1 Source to Target

6.1.1 sort.impv

```
func int[] bubbleSort(int[] a, int n){
  int i;
  int j;
  int tmp;
  printa(a);

  for (i = 0; i < n-1; i = i+1){
    for (j = 0; j < n-i-1; j = j+1){
      if (a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
        printa(a);
      }
    }
  }
  printa(a);
  return a;
```

```
}

func int[] selectionSort(int[] a, int n){
  int i;
  int j;
  int tmp;
  int min_idx;
  printa(a);

  for (i = 0; i < n-1; i = i+1){
    min_idx = i;

    for (j = i+1; j < n; j = j+1){
      if (a[j] < a[min_idx]){
        min_idx = j;
      }
    }

    tmp = a[min_idx];
    a[min_idx] = a[i];
    a[i] = tmp;
    printa(a);
  }
  printa(a);
  return a;
}

func int main() {
  int[] a;

  prints("BUBBLE SORT");
  a = [23, 11, 39, 44, 2, 16, 52];
  bubbleSort(a, 7);

  prints("SELECTION SORT");
  a = [23, 11, 39, 44, 2, 16, 52];
  selectionSort(a, 7);

  return 0;
```

```
}
```

```
BUBBLE SORT
[23, 11, 39, 44, 2, 16, 52]
[11, 23, 39, 44, 2, 16, 52]
[11, 23, 39, 2, 44, 16, 52]
[11, 23, 39, 2, 16, 44, 52]
[11, 23, 2, 39, 16, 44, 52]
[11, 23, 2, 16, 39, 44, 52]
[11, 2, 23, 16, 39, 44, 52]
[11, 2, 16, 23, 39, 44, 52]
[2, 11, 16, 23, 39, 44, 52]
[2, 11, 16, 23, 39, 44, 52]
SELECTION SORT
[23, 11, 39, 44, 2, 16, 52]
[2, 11, 39, 44, 23, 16, 52]
[2, 11, 39, 44, 23, 16, 52]
[2, 11, 16, 44, 23, 39, 52]
[2, 11, 16, 23, 44, 39, 52]
[2, 11, 16, 23, 39, 44, 52]
[2, 11, 16, 23, 39, 44, 52]
[2, 11, 16, 23, 39, 44, 52]
```

6.1.2 play-sort.impv

```
/* Generating sound of different sorting algorithms */

func note[] bubbleSort(int[] a, int n, note[] notes){
  int i;
  int j;
  int tmp;
  note[] tmparr;
  tmparr = convert(a, n);
  notes = append(notes, tmparr);

  for (i = 0; i < n-1; i = i+1){
    for (j = 0; j < n-i-1; j = j+1){
      if (a[j] > a[j+1]){
        tmp = a[j];
```

```
        a[j] = a[j+1];
        a[j+1] = tmp;
        tmparr = convert(a, n);
        notes = append(notes, tmparr);
      }
    }
  }
  tmparr = convert(a, n);
  notes = append(notes, tmparr);
  printNoteArr(notes);
  return notes;
}

func note[] selectionSort(int[] a, int n, note[] notes){
  int i;
  int j;
  int tmp;
  int min_idx;
  note[] tmparr;
  tmparr = convert(a, n);
  notes = append(notes, tmparr);

  for (i = 0; i < n-1; i = i+1){
    min_idx = i;

    for (j = i+1; j < n; j = j+1){
      if (a[j] < a[min_idx]){
        min_idx = j;
      }
    }

    tmp = a[min_idx];
    a[min_idx] = a[i];
    a[i] = tmp;
    tmparr = convert(a, n);
    notes = append(notes, tmparr);
  }
  tmparr = convert(a, n);
  notes = append(notes, tmparr);
```

```
  printNoteArr(notes);
  return notes;
}

func note[] convert(int[] a, int n){
  int i;
  note[] notes;
  note tmp;
  int val;

  notes = [<1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1,
"wh">];

  for(i = 0; i < n; i = i+1){
    tmp = <a[i]%5, "qr">;
    notes[i] = tmp;
  }

  return notes;
}


func int main() {
  int[] a;
  note[] bubble;
  note[] selection;

  prints("BUBBLE SORT");
  a = [54, 26, 11, 10, 32, 43];
  bubble = [<0, "qr">];
  render(bubbleSort(a, 6, bubble), "demo/bubble.mid", 13, 120);

  prints("SELECTION SORT");
  a = [54, 26, 11, 10, 32, 43];
  selection = [<0, "qr">];
  render(selectionSort(a, 6, selection), "demo/selection.mid", 13,
120);

  return 0;
```

```
}
```

```
BUBBLE SORT
[<0, qr>, <4, qr>, <1, qr>, <1, qr>, <0, qr>, <2, qr>, <3, qr>, <1,
qr>, <4, qr>, <1, qr>, <0, qr>, <2, qr>, <3, qr>, <1, qr>, <1, qr>,
<4, qr>, <0, qr>, <2, qr>, <3, qr>, <1, qr>, <1, qr>, <0, qr>, <4,
qr>, <2, qr>, <3, qr>, <1, qr>, <1, qr>, <0, qr>, <2, qr>, <4, qr>,
<3, qr>, <1, qr>, <1, qr>, <0, qr>, <2, qr>, <3, qr>, <4, qr>, <1,
qr>, <1, qr>, <0, qr>, <2, qr>, <3, qr>, <4, qr>, <1, qr>, <0, qr>,
<1, qr>, <2, qr>, <3, qr>, <4, qr>, <0, qr>, <1, qr>, <1, qr>, <2,
qr>, <3, qr>, <4, qr>, <0, qr>, <1, qr>, <1, qr>, <2, qr>, <3, qr>,
<4, qr>]
finished creating demo/bubble.mid!
SELECTION SORT
[<0, qr>, <4, qr>, <1, qr>, <1, qr>, <0, qr>, <2, qr>, <3, qr>, <0,
qr>, <1, qr>, <1, qr>, <4, qr>, <2, qr>, <3, qr>, <0, qr>, <1, qr>,
<1, qr>, <4, qr>, <2, qr>, <3, qr>, <0, qr>, <1, qr>, <1, qr>, <4,
qr>, <2, qr>, <3, qr>, <0, qr>, <1, qr>, <1, qr>, <2, qr>, <4, qr>,
<3, qr>, <0, qr>, <1, qr>, <1, qr>, <2, qr>, <3, qr>, <4, qr>, <0,
qr>, <1, qr>, <1, qr>, <2, qr>, <3, qr>, <4, qr>]
finished creating demo/selection.mid!
```

6.2 Test Suites

Tests were written iteratively so that every time a new feature was implemented, test cases were added to verify it was working. Two kinds of tests were written: ones intended to pass and ones intended to fail. We first began by writing simple tests such as for arithmetic operators, function calls, control flow statements, variable declarations, array operations, etc. After ensuring these test cases work, we could build on top of them and write more complex tests. This helped us keep track of what was broken as we continued to develop more features. After testing all our features, we were able to include more long and complex programs such as our sorting demo.

```
test-arith...OK
test-arr-access...OK
test-arr-append-notes...OK
test-arr-assign...OK
test-arr...OK
```

```
test-convert...OK
test-demo-sort...OK
test-fib...OK
test-for...OK
test-gcd-recursive...OK
test-gcd...OK
test-if...OK
test-mcd...OK
test-note...OK
test-note2...OK
test-note3...OK
test-printa...OK
test-printbig...OK
test-printi...OK
test-printmidi...OK
test-printn...OK
test-prints...OK
test-render...OK
test-sort...OK
test-var-int...OK
test-var-string...OK
test-while...OK
fail-duplicate-function...OK
fail-inconsistent-arr-type...OK
fail-inconsistent-return-type...OK
fail-rhythm...OK
fail-tonetype...OK
fail-undeclared-identifier...OK
fail-undefined-function...OK
```

6.3 Test Automation

To quickly create a new test, we wrote a script createTest.sh to generate a blank template for a .impv file and a corresponding .out or .err file for the test name. To run all of the tests, we wrote a test script testAll.sh which runs every test and checks the output/error message matches the expected output/error message. To run an individual test, we run the testAll.sh with the individual test file name. The detailed log messages for running the tests are dumped in the testall.log file.

If the test passes, then it is marked with an OK. Otherwise, it is indicated by a FAILURE and the error message can be found in the testall.log. If the test failed due to a difference in the expected output, the differences are found in a separate .diff file and the actual output is directed into a new .out file.

7. Lessons Learned

Emily

Learning: Functional programming is pretty cool. Learnt a lot about how compilers work, how to iteratively test your code, and how to effectively work on a large group coding project.
Advice: Take advantage of office hours early on!!! Especially if you feel lost on how to get started with something. Don't be afraid to ask for guidance or help. Over communicate with your team so that you are all on the same page.

Natalia

Learning: Did not know much about functional programming before this course, and now can most definitely say I know quite a bit. This was a very interesting class for me since I am coming at it as an Operations Research major (who has an interest in taking quite a few CS classes). Learned a lot about implementation and how programming languages work as well as how to work with a team and make the most of it! Also! Do not worry that your project is not enough - start small,m set goals, and build up to them.
Advice: Tactical -> read the advice section of a few final reports -> plan early and understand what you are doing when you start -> seek help from your teammates -> understand lecture (and the jokes) -> seek help from TAs when needed *not right before the deadline though because those office are CRAZY* -> get to know your teammates and classmates (this is college after all and is a great place to meet other smart people) -> work consistently on code BUT think and talk it out with a partner before you do it -> aaaaand just know you are here for a long time AND a good time

Josh

Learning: How a programming language goes from source code to executable and the implementation of the intermediate steps involved.
Advice: Start the project early.

Alice

Learning: A lot about the different components of a compiler—scanner, parser, AST,

semantics, codegen, all of it and how they connect.
Advice: Start the project early. Go to office hours. Pay attention during the codegen lecture.

8. Appendix

Everything we wrote…

parser.mly

```
/* Authors: Emily Li, Natalia Dorogi, Alice Zhang */

%{ open Ast %}

%token SEP EOF ENDLINE

%token ASSIGN
%token LPAREN RPAREN LBRACK RBRACK LCURLY RCURLY
%token COMMA

%token PLUS MINUS TIMES DIVIDE MOD
%token EQ NEQ LT LTE GT GTE
%token AND OR NOT

%token NOTE
%token INT BOOL STRING NONE

%token FUNC IN IF ELSE FOR WHILE RETURN

%token <bool> LIT_BOOL
%token <int> LIT_INT
%token <string> LIT_STRING
%token <string> ID

%nonassoc NOELSE
%nonassoc ELSE
%left SEP
%nonassoc ASSIGN

%left OR
%left AND
%left EQ NEQ
%nonassoc LT LTE GT GTE
%left COMMA
%left PLUS MINUS
%left TIMES DIVIDE MOD
```

```
%nonassoc NOT

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

fdecl:
  | FUNC typ ID LPAREN params_opt RPAREN LCURLY vdecl_list stmt_list RCURLY
    { { ftype = $2;
        fname = $3;
        params = List.rev $5;
        vars = List.rev $8;
        body = List.rev $9 } }

params_opt:
  | /* nothing */ { [] }
  | params_list { $1 }

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

typ:
  | BOOL    { Bool }
  | INT     { Int }
  | STRING  { String }
  | NOTE    { Note }
  | NONE    { None }
  | typ LBRACK RBRACK { Array($1) }

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

vdecl:
  | typ ID SEP { ($1, $2) }
```

```
stmt_list:
  | stmt { [$1] }
  | stmt_list stmt { $2 :: $1 }

stmt:
  | expr SEP                               { Expr $1              }
  | RETURN expr_opt SEP                    { Return $2            }
  | LCURLY stmt_list RCURLY                { 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 SEP expr SEP expr_opt RPAREN stmt
                                         { For($3, $5, $7, $9)   }
  | WHILE LPAREN expr RPAREN stmt                      { While($3, $5)        }

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

expr:
  /* literals */
  | literals { $1 }
  | ID { Id($1) }

  /* arithmetic operations */
  | expr PLUS   expr { Binop($1, Add, $3) }
  | expr MINUS  expr { Binop($1, Sub, $3) }
  | expr TIMES  expr { Binop($1, Mul, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr MOD expr    { Binop($1, Mod, $3) }

  /* boolean operations */
  | expr EQ  expr { Binop($1, Eq,  $3) }
  | expr NEQ expr { Binop($1, Neq, $3) }
  | expr LT  expr { Binop($1, Lt,  $3) }
  | expr LTE expr { Binop($1, Lte, $3) }
  | expr GT  expr { Binop($3, Lt,  $1) }
  | expr GTE expr { Binop($3, Lte, $1) }
  |      NOT expr { Uniop(Not, $2) }
  | expr AND expr { Binop($1, And, $3) }
  | expr OR  expr { Binop($1, Or,  $3) }

  /* variable assignment */
  | ID ASSIGN expr { Assign($1, $3) }

  /* function call */
  | ID LPAREN args_opt RPAREN { Call($1, $3) }
```

```
  /* array access, assign, append */
  | ID LBRACK expr RBRACK { ArrayAccess($1, $3) }
  | ID LBRACK expr RBRACK ASSIGN expr { ArrayAssign($1, $3, $6) }

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

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

literals:
  | LIT_BOOL         { LitBool($1) }
  | LIT_INT          { LitInt($1) }
  | LIT_STRING       { LitString($1) }
  | lit_note         { $1 }
  | lit_array        { $1 }

lit_note:
  | LT expr COMMA LIT_STRING GT  { LitNote($2, $4) }

lit_array:
  | LBRACK items_list RBRACK { LitArray($2) }

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

scanner.mll

```
(* Authors: Emily Li, Natalia Dorogi, Josh Choi *)

{ open Parser }

let lowercase = ['a'-'z']
let uppercase = ['A'-'Z']
let letter = lowercase | uppercase
let digit = ['0'-'9']
let keys = ("C" | "C#" | "D" | "D#" | "E" | "F" | "F#" | "G" | "G#" | "A" | "A#" |
"B")

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }
| ';' { SEP }
| eof { EOF }
```

```
(* SCOPING *)
| '(' { LPAREN }
| ')' { RPAREN }
| '[' { LBRACK }
| ']' { RBRACK }
| '{' { LCURLY }
| '}' { RCURLY }
(* OPERATORS *)
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| '%' { MOD }
| '=' { ASSIGN }
| "==" { EQ }
| "!=" { NEQ }
| '<' { LT }
| "<=" { LTE }
| '>' { GT }
| ">=" { GTE }
| ',' { COMMA }
(* KEYWORDS *)
(* DATA TYPES *)
| "note" { NOTE }
| "int" { INT }
| "bool" { BOOL }
| "string" { STRING }
| "none" { NONE }
(* BOOLEAN LOGIC *)
| "and" { AND }
| "or" { OR }
| "not" { NOT }
(* PROGRAM STRUCTURE *)
| "func" { FUNC }
| "in" { IN }
| "if" { IF }
| "else" { ELSE }
| "for" { FOR }
| "while" { WHILE }
| "return" { RETURN }
(* LITERALS *)
| '"' (('\\' '"'| [^'"'])* as str) '"' { LIT_STRING(str) }
| ['0'-'9']+ as lit { LIT_INT(int_of_string lit) }
| "true"   { LIT_BOOL(true)  }
| "false"  { LIT_BOOL(false) }
(* IDENTIFIERS *)
| (lowercase | '_') (letter | digit | '_')* as lit { ID(lit) }
```

```
(* COMMENTS *)
| "//" { commentLine lexbuf }
| "/*" { commentBlock 0 lexbuf }

and commentLine = parse
| ('\n' | '\r' | "\r\n" | eof) { ENDLINE }
| _ { commentLine lexbuf }

and commentBlock level = parse
| "/*" { commentBlock (level + 1) lexbuf }
| "*/" { if level == 0 then token lexbuf
        else commentBlock (level - 1) lexbuf }
| _ { commentBlock level lexbuf }
```

ast.ml

```
(* Authors: Alice Zhang, Josh Choi *)

type op = Add | Sub | Mul | Div | Mod |
          Eq | Neq | Lt | Lte | And | Or

type uop = Not

type typ = Bool | Int | String | Note | Tone | Rhythm | None |
           Array of typ

type bind = typ * string

type expr =
    LitBool of bool
  | LitInt of int
  | LitString of string
  | LitNote of expr * string
  | LitArray of expr list
  | Id of string
  | Binop of expr * op * expr
  | Uniop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | ArrayAccess of string * expr
  | ArrayAssign of string * expr * 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 = {
    ftype : typ;
    fname : string;
    params : bind list;
    vars : bind list;
    body : stmt list;
}

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mul -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Eq -> "=="
  | Neq -> "!="
  | Lt -> "<"
  | Lte -> "<="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Not -> "!"

let rec string_of_expr = function
    LitBool(true) -> "true"
  | LitBool(false) -> "false"
  | LitInt(i) -> string_of_int i
  | LitString(s) -> s
  | LitNote(t, r) -> "<" ^ string_of_expr t ^ ", " ^ r ^ ">"
  | LitArray(el) -> "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Uniop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | ArrayAccess(s, e) -> s ^ "[" ^ string_of_expr e ^ "]"
```

```
  | ArrayAssign(v, l, e) -> v ^ "[" ^ string_of_expr l ^ "]" ^ " = " ^
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 rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | String -> "string"
  | Note -> "note"
  | Tone -> "tone"
  | Rhythm -> "rhythm"
  | None -> "none"
  | Array(t) -> string_of_typ t ^ "[]"

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

let string_of_fdecl fdecl =
  string_of_typ fdecl.ftype ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.params) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.vars) ^
  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)
```

semant.ml

```
(* Authors: Emily Li, Natalia Dorogi, Alice Zhang *)

open Ast
```

```
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.
   Check each global variable, then check each function *)

let check (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 (None, 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, ty) = StringMap.add name {
      ftype = None;
      fname = name;
      params = [(ty, "x")];
      vars = [];
      body = [] } map
    in List.fold_left add_bind StringMap.empty [ ("print", Int); ("printi", Int);
("prints", String); ("printn", Note); ("printbig", Int); ("printmidi", String);
("printa", Array(Int)); ("printNoteArr", Array(Note))];
  in

  let built_in_decls =
    StringMap.add "render" {
      ftype = None;
      fname = "render";
      params = [(Array(Note), "noteArr"); (String, "filename"); (Int, "key"); (Int,
"tempo")]; (* formals *)
```

```
    vars = []; (* locals *)
    body = [] } built_in_decls
in

let built_in_decls =
  StringMap.add "append" {
    ftype = Array(Note);
    fname = "append";
    params = [(Array(Note), "a1"); (Array(Note), "a2")]; (* formals *)
    vars = []; (* locals *)
    body = [] } built_in_decls
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 (*adds function to 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 params or vars are void or duplicates *)
  check_binds "params" func.params;
  check_binds "vars" func.vars;

  (* 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
```

```
(* Build var 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.params @ func.vars )
in

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

(* Raise exception if given tone is not valid (only 1-6) *)
let check_tone = function
  (Int, t) -> (Int, t)
  | _ -> raise (Failure ("invalid tone assignment, must be int expression"))
in

(* Raise exception if given rhythm is not valid *)
let check_rhythm r =
  match r with
    | "wh" -> "0"
    | "hf" -> "1"
    | "qr" -> "2"
    | "ei" -> "3"
    | "sx" -> "4"
    | _ -> raise (Failure ("invalid rhythm assignment " ^ r))
  in

let get_array_type = function
    Array(t) -> t
  | _ -> raise (Failure "invalid array type")
in

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
    LitInt  l -> (Int, SLitInt l)
  | LitString l -> (String, SLitString l)
  | LitBool l  -> (Bool, SLitBool l)
  | LitNote (t, r) ->
    let (typ, t') = expr t in
    (Note, SLitNote(check_tone(typ, t'), check_rhythm r))
  | LitArray l -> let array = List.map expr l in
      let rec type_check = function
        (t1, _) :: [] -> (Array(t1), SLitArray(array))
        | ((t1,_) :: (t2,_) :: _) when t1 != t2 ->
          raise (Failure ("inconsistent array types, expected " ^ string_of_typ
```

```
t1 ^ " but found " ^ string_of_typ t2))
            | _ :: t -> type_check t
            | [] -> raise (Failure ("empty array"))
            in type_check array
      | ArrayAccess (a, i) ->
        let (t, i') = expr i in
        let ty = (type_of_identifier a) in
        (get_array_type ty, SArrayAccess (a, (t, i')))
      | ArrayAssign(v, i, e) ->
        let (t, i') = expr i in
        let (te, e') = expr e in
        let ty = (type_of_identifier v) in
        (get_array_type ty, SArrayAssign (v, (t, i'), (te, e')))
      | NoExpr     -> (None, 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')))
      | Uniop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
          | Not when t = Bool -> Bool
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUniop(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 | Mul | Div | Mod when same && t1 = Int  -> Int
          | Eq | Neq            when same               -> Bool
          | Lt | Lte
                     when same && (t1 = Int) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | _ -> raise (
            Failure ("illegal binary operator " ^
                       string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                       string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | Call(fname, args) as call ->
```

```
      let fd = find_func fname in
      let param_length = List.length fd.params 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.params args
      in (fd.ftype, SCall(fname, args'))
in

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

(* Return a semantically-checked statement i.e. containing sexprs *)
let rec check_stmt = function
    Expr e -> SExpr (expr e) (*check the expr semantically woo*)
  | 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.ftype then SReturn (t, e')
    else raise (
     Failure ("return gives " ^ string_of_typ t ^ " expected " ^
            string_of_typ func.ftype ^ " 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 *)
{ sftype = func.ftype;
```

```
      sfname = func.fname;
      sparams = func.params;
      svars  = func.vars;
      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)
```

codegen.ml

```
(* Authors: Alice Zhang, Emily Li *)

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 "Improv" 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 string_t   = L.pointer_type (L.i8_type context)
  and void_t     = L.void_type   context in

  (* Return the LLVM type for an Improv type *)
  let rec ltype_of_typ = function
      A.Int  -> i32_t
    | A.String  -> string_t
    | A.Bool  -> i1_t
    | A.Tone  -> i32_t
    | A.Rhythm  -> string_t
    | A.Note  -> L.struct_type context [| i32_t ; string_t |]
    | A.Array(t)  -> L.struct_type context [| i32_t ; L.pointer_type (ltype_of_typ
t) |]
    | A.None  -> void_t
  in
```

```
  let rec int_range = function
      0 -> [ ]
    | 1 -> [ 0 ]
    | n -> int_range (n - 1) @ [ n - 1 ] 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.String -> L.const_string context ""
        | _ -> L.const_int (ltype_of_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 printn_t : L.lltype =
    L.function_type i32_t [| ltype_of_typ(A.Note) |] in
  let printn_func : L.llvalue =
    L.declare_function "printn" printn_t the_module in

  let printa_t : L.lltype =
    L.function_type i32_t [| ltype_of_typ(A.Array(Int)) |] in
  let printa_func : L.llvalue =
    L.declare_function "printa" printa_t the_module in

  let render_t : L.lltype =
    L.function_type i32_t [| ltype_of_typ(A.Array(Note)) ; string_t ; i32_t ; i32_t
|] in
  let render_func : L.llvalue =
    L.declare_function "render" render_t the_module in

  let printmidi_t : L.lltype =
    L.function_type i32_t [| string_t |] in
  let printmidi_func : L.llvalue =
    L.declare_function "printmidi" printmidi_t the_module in

  let append_t : L.lltype =
    L.function_type (ltype_of_typ(A.Array(Note))) [| ltype_of_typ(A.Array(Note)) ;
```

```
ltype_of_typ(A.Array(Note)) |] in
  let append_func : L.llvalue =
    L.declare_function "append" append_t the_module in

  let printNoteArr_t : L.lltype =
    L.function_type i32_t [| ltype_of_typ(A.Array(Note)) |] in
  let printNoteArr_func : L.llvalue =
    L.declare_function "printNoteArr" printNoteArr_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 param_types =
            Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sparams)
      in let ftype = L.function_type (ltype_of_typ fdecl.sftype) param_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 string_format_str = L.build_global_stringptr "%s\n" "fmt" builder in

    (* Construct the function's "locals": param 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_param 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 params = List.fold_left2 add_param StringMap.empty fdecl.sparams
        (Array.to_list (L.params the_function)) in
```

```
      List.fold_left add_local params fdecl.svars
    in

    (* Return the value for a variable or param 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

    let build_array len el =
      let arr_mem = L.build_array_malloc (L.type_of (List.hd el)) len "tmp" builder
in
        List.iter (fun idx ->
          let arr_ptr = (L.build_gep arr_mem [| L.const_int i32_t idx |] "tmp2"
builder) in
          let e_val = List.nth el idx in
          ignore (L.build_store e_val arr_ptr builder)
        ) (int_range (List.length el));
        let len_arr_ptr = L.struct_type context [| i32_t ; L.pointer_type
(L.type_of (List.hd el)) |] in
        let struc_ptr = L.build_malloc len_arr_ptr "arr_literal" builder in
        let first_store = L.build_struct_gep struc_ptr 0 "first" builder in
        let second_store = L.build_struct_gep struc_ptr 1 "second" builder in
        ignore (L.build_store len first_store builder);
        ignore (L.build_store arr_mem second_store builder);
        let result = L.build_load struc_ptr "actual_arr_literal" builder in
      result
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) = match e with
             SLitInt i -> L.const_int i32_t i
      | SLitString s -> L.build_global_stringptr s "str" builder
      | SLitBool b  -> L.const_int i1_t (if b then 1 else 0)

      (* note struct type *)
      | SLitNote (i, s) ->
        let i'= expr builder i in
        let s' = L.build_global_stringptr s "str" builder in
        let struc =  L.struct_type context [| i32_t ; string_t |] in
        let struct_ptr = L.build_malloc struc "struct_mem" builder in
        let first_store = L.build_struct_gep struct_ptr 0 "tone" builder in
        let second_store = L.build_struct_gep struct_ptr 1 "rhythm" builder in
        ignore (L.build_store i' first_store builder);
        ignore (L.build_store s' second_store builder);
        let result = L.build_load struct_ptr "struct_literal" builder in result
```

```
      | SLitArray l  -> let len = L.const_int i32_t (List.length l) in
                        let el = List.map (fun e' -> expr builder e') (List.rev l)
in
                        build_array len el
      | SArrayAccess(s, ind1) ->
        let i' = expr builder ind1 in
        let v' = L.build_load (lookup s) s builder in
        let extract_value = L.build_extractvalue v' 1 "extract_value" builder in
        let extract_array = L.build_gep extract_value [| i' |] "extract_array"
builder in
        let result = L.build_load extract_array s builder in result

      | SArrayAssign(v, i, e) -> let e' = expr builder e in
        let i' = expr builder i in
        let v' = L.build_load (lookup v) v builder in
        let extract_value = L.build_extractvalue v' 1 "extract_value" builder in
        let extract_array = L.build_gep extract_value [| i' |] "extract_array"
builder in
        ignore (L.build_store e' extract_array builder); e'

      | 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 (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.Mul     -> L.build_mul
        | A.Div     -> L.build_sdiv
        | A.Mod     -> L.build_srem
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Eq      -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Lt      -> L.build_icmp L.Icmp.Slt
        | A.Lte     -> L.build_icmp L.Icmp.Sle
        ) e1' e2' "tmp" builder
      | SUniop(op, ((t, _) as e)) ->
        let e' = expr builder e in
        (match op with
        | A.Not                   -> L.build_not) e' "tmp" builder
      | SCall ("print", [e])
      | SCall ("printbig", [e]) ->
        L.build_call printbig_func [| (expr builder e) |] "printbig" builder
```

```
      | SCall ("printi", [e]) ->
        L.build_call printf_func [| int_format_str ; (expr builder e) |]
          "printf" builder
      | SCall ("prints", [e]) ->
            L.build_call printf_func [| string_format_str ; (expr builder e) |]
              "printf" builder
      | SCall ("printn", [e]) ->
        L.build_call printn_func [| (expr builder e) |]
          "printn" builder
      | SCall ("printa", [e]) ->
        L.build_call printa_func [| (expr builder e) |]
          "printa" builder
      | SCall ("render", [arr ; file ; key ; tempo]) ->
        L.build_call render_func [| (expr builder arr) ; (expr builder file) ;
(expr builder key) ; (expr builder tempo) |]
          "render" builder
      | SCall ("printmidi", [e]) ->
        L.build_call printmidi_func [| (expr builder e) |]
          "printmidi" builder
      | SCall ("append", [a1; a2]) ->
        L.build_call append_func [| (expr builder a1) ; (expr builder a2) |]
          "append" builder
      | SCall ("printNoteArr", [a1]) ->
        L.build_call printNoteArr_func [| (expr builder a1) |]
          "printNoteArr" 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.sftype with
                          A.None -> ""
                        | _ -> 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.sftype with
                        (* Special "return nothing" instr *)
                        A.None -> 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.sftype with
        A.None -> L.build_ret_void
      | A.String -> L.build_ret (L.const_string context "")
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

improv.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef  __APPLE__
#include <malloc.h>
#endif
#include "midifile.h"
#include "improv.h"

/* print note literal */
void printn(Note note){
  char *rhythm_map[6] ={"wh", "hf", "qr", "ei", "sx"};
  printf("<%d, %s>\n", note.tone, rhythm_map[atoi(note.rhythm)]);
}

void printa(int len, int *arr){
  int i;
  printf("[");
  for(i = 0; i < len; i++, arr++){
    if(i == len-1){
      printf("%d", *arr);
    } else{
      printf("%d, ", *arr);
    }
  }
  printf("]\n");
}

int* getKey(int key){
  switch(key){
    case CMAJ:
      return cmaj;
    case CSHARPMAJ:
      return csharpmaj;
```

```
    case DMAJ:
      return dmaj;
    case DSHARPMAJ:
      return dsharpmaj;
    case EMAJ:
      return emaj;
    case FMAJ:
      return fmaj;
    case FSHARPMAJ:
      return fsharpmaj;
    case GMAJ:
      return gmaj;
    case GSHARPMAJ:
      return gsharpmaj;
    case AMAJ:
      return amaj;
    case ASHARPMAJ:
      return asharpmaj;
    case BMAJ:
      return bmaj;
    case CMIN:
      return cmin;
    case CSHARPMIN:
      return csharpmin;
    case DMIN:
      return dmin;
    case DSHARPMIN:
      return dsharpmin;
    case EMIN:
      return emin;
    case FMIN:
      return fminor;
    case FSHARPMIN:
      return fsharpmin;
    case GMIN:
      return gmin;
    case GSHARPMIN:
      return gsharpmin;
    case AMIN:
      return amin;
    case ASHARPMIN:
      return asharpmin;
    case BMIN:
      return bmin;
  }
}
```

```
/* create midi file */
void render_backend(Note* notes, int size, char* filename, int key[], int tempo){
  MIDI_FILE *mf;
  int i;

  int rhythms[] = {MIDI_NOTE_BREVE, MIDI_NOTE_MINIM, MIDI_NOTE_CROCHET,
MIDI_NOTE_QUAVER, MIDI_NOTE_SEMIQUAVER};

  if ((mf = midiFileCreate(filename, TRUE))){
             midiSongAddTempo(mf, 1, tempo);
             midiFileSetTracksDefaultChannel(mf, 1, MIDI_CHANNEL_1);
             midiTrackAddProgramChange(mf, 1, MIDI_PATCH_ELECTRIC_GUITAR_JAZZ);
             midiSongAddSimpleTimeSig(mf, 1, 4, MIDI_NOTE_CROCHET);

    for(i = 0; i < size; i++, notes++){
      /* printn(*notes); */
      midiTrackAddNote(mf, 1, key[notes->tone], rhythms[atoi(notes->rhythm)],
MIDI_VOL_HALF, TRUE, FALSE);
    }

             midiFileClose(mf);
    printf("finished creating %s!\n", filename);
       }
}

void render(Note_Arr noteArr, char* filename, int key, int tempo){
  int *keyNotes = getKey(key);
  render_backend(noteArr.arr, noteArr.len, filename, keyNotes, tempo);
}

void printmidi(char* filename){
  TestEventList(filename);
}

void printnInternal(Note note){
  char *rhythm_map[6] ={"wh", "hf", "qr", "ei", "sx"};
  printf("<%d, %s>", note.tone, rhythm_map[atoi(note.rhythm)]);
}

void printNoteArr(Note_Arr a){
  int i;
  printf("[");
  for(i = 0; i < a.len; i++, a.arr++){
    if(i == a.len-1){
      printnInternal(*a.arr);
    } else{
      printnInternal(*a.arr);
```

```
      printf(", ");
    }
  }
  printf("]\n");
}

Note_Arr append(Note_Arr a1, Note_Arr a2){
  int i;
  Note_Arr result;
  result.len = a1.len + a2.len;
  result.arr = malloc(result.len * sizeof(Note));
  void *ptr = result.arr;

  int sizeofa1 = (a1.len)*sizeof(Note);
  int sizeofa2 = (a2.len)*sizeof(Note);

  memcpy(ptr, a1.arr, sizeofa1);
  memcpy(ptr+sizeofa1, a2.arr, sizeofa2);

  return result;
}
```

improv.h

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

typedef struct Note{
  int tone;
  char *rhythm;
} Note;

typedef struct Note_Arr {
  int len;
  Note *arr;
} Note_Arr;

typedef struct Arr {
  int len;
  void *arr;
} Arr;

#define CMAJ 1
#define CSHARPMAJ 2
#define DMAJ 3
#define DSHARPMAJ 4
```

```
#define EMAJ 5
#define FMAJ 6
#define FSHARPMAJ 7
#define GMAJ 8
#define GSHARPMAJ 9
#define AMAJ 10
#define ASHARPMAJ 11
#define BMAJ 12
#define CMIN 13
#define CSHARPMIN 14
#define DMIN 15
#define DSHARPMIN 16
#define EMIN 17
#define FMIN 18
#define FSHARPMIN 19
#define GMIN 20
#define GSHARPMIN 21
#define AMIN 22
#define ASHARPMIN 23
#define BMIN 24

// Major Pentatonic
int cmaj[] = {MIDI_OCTAVE_5, MIDI_OCTAVE_5 + MIDI_NOTE_D,
            MIDI_OCTAVE_5 + MIDI_NOTE_E, MIDI_OCTAVE_5 + MIDI_NOTE_G,
            MIDI_OCTAVE_5 + MIDI_NOTE_A};
int csharpmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP, MIDI_OCTAVE_5 +
MIDI_NOTE_D_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_F, MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP};
int dmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_D, MIDI_OCTAVE_5 + MIDI_NOTE_E,
            MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_A,
            MIDI_OCTAVE_5 + MIDI_NOTE_B};
int dsharpmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_F,
            MIDI_OCTAVE_5 + MIDI_NOTE_G, MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_C};
int emaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_E, MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_B,
            MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP};
int fmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_F, MIDI_OCTAVE_5 + MIDI_NOTE_G,
            MIDI_OCTAVE_5 + MIDI_NOTE_A, MIDI_OCTAVE_5 + MIDI_NOTE_C,
            MIDI_OCTAVE_5 + MIDI_NOTE_D};
int fsharpmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP, MIDI_OCTAVE_5 +
MIDI_NOTE_G_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP};
int gmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_G, MIDI_OCTAVE_5 + MIDI_NOTE_A,
            MIDI_OCTAVE_5 + MIDI_NOTE_B, MIDI_OCTAVE_5 + MIDI_NOTE_D,
```

```
            MIDI_OCTAVE_5 + MIDI_NOTE_E};
int gsharpmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP, MIDI_OCTAVE_5 +
MIDI_NOTE_A_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_C, MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_F};
int amaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_A, MIDI_OCTAVE_5 + MIDI_NOTE_B,
            MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_E,
            MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP};
int asharpmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_C,
            MIDI_OCTAVE_5 + MIDI_NOTE_D, MIDI_OCTAVE_5 + MIDI_NOTE_F,
            MIDI_OCTAVE_5 + MIDI_NOTE_G};
int bmaj[] = {MIDI_OCTAVE_5 + MIDI_NOTE_B, MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP};

// Minor Pentatonic
int cmin[] = { MIDI_OCTAVE_5, MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_F, MIDI_OCTAVE_5 + MIDI_NOTE_G,
            MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP};
int csharpmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_E,
            MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_B};
int dmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_D, MIDI_OCTAVE_5 + MIDI_NOTE_F,
            MIDI_OCTAVE_5 + MIDI_NOTE_G, MIDI_OCTAVE_5 + MIDI_NOTE_A,
            MIDI_OCTAVE_5 + MIDI_NOTE_C};
int dsharpmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP, MIDI_OCTAVE_5 +
MIDI_NOTE_F_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP};
int emin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_E, MIDI_OCTAVE_5 + MIDI_NOTE_G,
            MIDI_OCTAVE_5 + MIDI_NOTE_A, MIDI_OCTAVE_5 + MIDI_NOTE_B,
            MIDI_OCTAVE_5 + MIDI_NOTE_D};
int fminor[] = { MIDI_OCTAVE_5 + MIDI_NOTE_F, MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_C,
            MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP};
int fsharpmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_A,
            MIDI_OCTAVE_5 + MIDI_NOTE_B, MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_E};
int gmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_G, MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_C, MIDI_OCTAVE_5 + MIDI_NOTE_D,
            MIDI_OCTAVE_5 + MIDI_NOTE_F};
int gsharpmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_B,
            MIDI_OCTAVE_5 + MIDI_NOTE_C_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP};
int amin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_A, MIDI_OCTAVE_5 + MIDI_NOTE_C,
            MIDI_OCTAVE_5 + MIDI_NOTE_D, MIDI_OCTAVE_5 + MIDI_NOTE_E,
            MIDI_OCTAVE_5 + MIDI_NOTE_G};
```

```
int asharpmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_A_SHARP, MIDI_OCTAVE_5 +
MIDI_NOTE_C_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_D_SHARP, MIDI_OCTAVE_5 + MIDI_NOTE_F,
            MIDI_OCTAVE_5 + MIDI_NOTE_G_SHARP};
int bmin[] = { MIDI_OCTAVE_5 + MIDI_NOTE_B, MIDI_OCTAVE_5 + MIDI_NOTE_D,
            MIDI_OCTAVE_5 + MIDI_NOTE_E, MIDI_OCTAVE_5 + MIDI_NOTE_F_SHARP,
            MIDI_OCTAVE_5 + MIDI_NOTE_A};
```

improv.ml

```
(* Top-level of the improv 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: ./improv.native [-a|-s|-l|-c] [file.impv]" 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)
```

Makefile

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

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

# "make microc.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

improv.native : improv.ml codegen.ml
	ocamlbuild -use-ocamlfind improv.native -pkgs llvm,llvm.analysis

parser.native : parser.mly ast.mli scanner.mll
	ocamlbuild parser.native

scanner.native : scanner.mll
	ocamlbuild scanner.native

# "make clean" removes all generated files
.PHONY : clean
clean :
	rm -f improv.native
	rm -f hello.lli
	rm -rf _build
	rm -f *.o

.PHONY : all
all : clean improv.native midifile.o

# midifile.o:midifile.c   midifile.h
# midiutil.o:midiutil.c   midiutil.h

# improv : improv.c midifile.o
#	gcc -Wall -DBUILD_TEST midifile.o improv.c -o improv
# cc -o improv -DBUILD_TEST improv.c


# Building the tarball

TESTS = \
#  add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2 func1 \
#  func2 func3 func4 func5 func6 func7 func8 func9 gcd2 gcd global1 \
#  global2 global3 hello if1 if2 if3 if4 if5 if6 local1 local2 ops1 \
```

```
#  ops2 printbig var1 var2 while1 while2

FAILS = \
#  assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1 float2 \
#  for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 \
#  func8 func9 global1 global2 if1 if2 if3 nomain printbig printb print \
#  return1 return2 while1 while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
          $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

# testall.sh
TARFILES = ast.ml sast.ml codegen.ml Makefile _tags improv.ml parser.mly \
       README scanner.mll semant.ml \
       Dockerfile \
       $(TESTFILES:%=tests/%)

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

Demo - Bubble and Selection Sort

`sort.impv`

```
/* Generating sound of different sorting algorithms */

func note[] bubbleSort(int[] a, int n, note[] notes){
  int i;
  int j;
  int tmp;
  note[] tmparr;
  tmparr = convert(a, n);
  notes = append(notes, tmparr);

  for (i = 0; i < n-1; i = i+1){
    for (j = 0; j < n-i-1; j = j+1){
      if (a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
        tmparr = convert(a, n);
        notes = append(notes, tmparr);
      }
    }
  }
  tmparr = convert(a, n);
```

```
  notes = append(notes, tmparr);
  printNoteArr(notes);
  printa(a);
  return notes;
}

func note[] selectionSort(int[] a, int n, note[] notes){
  int i;
  int j;
  int tmp;
  int min_idx;
  note[] tmparr;
  tmparr = convert(a, n);
  notes = append(notes, tmparr);

  printa(a);

  for (i = 0; i < n-1; i = i+1){
    min_idx = i;

    for (j = i+1; j < n; j = j+1){
      if (a[j] < a[min_idx]){
        min_idx = j;
      }
    }

    tmp = a[min_idx];
    a[min_idx] = a[i];
    a[i] = tmp;
    tmparr = convert(a, n);
    notes = append(notes, tmparr);
    printa(a);
  }
  tmparr = convert(a, n);
  notes = append(notes, tmparr);
  printNoteArr(notes);
  printa(a);
  return notes;
}

func note[] convert(int[] a, int n){
  int i;
  note[] notes;
  note tmp;
  int val;

  notes = [<1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">];
```

```
  for(i = 0; i < n; i = i+1){
    tmp = <a[i]%5, "qr">;
    notes[i] = tmp;
  }

  return notes;
}

func int main() {
  int[] a;
  note[] bubble;
  note[] selection;

  prints("BUBBLE SORT");
  a = [54, 26, 11, 10, 32, 43];
  bubble = [<0, "qr">];
  render(bubbleSort(a, 6, bubble), "demo/bubble.mid", 13, 120);

  prints("SELECTION SORT");
  a = [54, 26, 11, 10, 32, 43];
  selection = [<0, "qr">];
  render(selectionSort(a, 6, selection), "demo/selection.mid", 13, 120);

  return 0;
}
```

Demo - Ole McDonald

`mcd.impv`

```
func int main() {
  note[] arr;
  arr = [<1, "qr">, <1, "qr">, <1, "qr">, <4, "qr">,
    <0, "qr">, <0, "qr">, <4, "hf">, <3, "qr">,
    <3, "qr">, <2, "qr">, <2, "qr">, <1, "hf">];
  render(arr, "demo/mcd.mid", 13, 120);
  return 0;
}
```

Test Suite

`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 microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
IMPROV="./src/improv.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] [.impv 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/.impv//'`
    reffile=`echo $1 | sed 's/.impv$//'`
    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
${basename}.out" &&
    Run "$IMPROV" "$1" ">" "${basename}.ll" &&
```

```
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "src/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
}

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$IMPROV" "<" $1 "2>" "${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 [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.impv tests/fail-*.impv"
fi

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

```
exit $globalerror
```

test-arith.impv

```
func int add(int x, int y) {
  return x + y;
}

func int sub(int x, int y) {
  return x - y;
}

func int mul(int x, int y) {
  return x * y;
}

func int div(int x, int y) {
  return x / y;
}

func int mod(int x, int y) {
  return x % y;
}

func int main() {
  printi(add(4115, 42));
  printi(sub(4115, 42));
  printi(mul(4115, 42));
  printi(div(4115, 42));
  printi(mod(4115, 42));
  return 0;
}
```

test-arr-access.impv

```
func int main() {
  int[] int_arr;
  note[] note_arr;

  int_arr = [1, 2, 3, 4, 5];
  printi(int_arr[1]);

  note_arr = [<1, "wh">, <5, "wh">];
  printn(note_arr[1]);
```

```
  return 0;
}
```

test-arr-append-notes.impv

```
func int main() {
  note[] n1;
  note[] n2;
  note[] n3;

  n1 = [<1, "wh">, <5, "wh">];
  n2 = [<4, "wh">];

  n1 = append(n1, n2);
  printNoteArr(n1);

  n1 = append(n1, [<2, "hf">]);
  printNoteArr(n1);

  return 0;
}
```

test-arr-assign.impv

```
func int main() {
  int[] int_arr;
  note[] note_arr;

  int_arr = [1, 2, 3, 4, 5];
  int_arr[1] = 10;
  printi(int_arr[1]);

  note_arr = [<1, "wh">, <5, "wh">];
  note_arr[0] = <2, "hf">;
  printn(note_arr[0]);

  return 0;
}
```

test-arr.impv

```
func int main() {
  int[] int_arr;
  note[] note_arr;

  int_arr = [1, 2];
```

```
  note_arr = [<1, "wh">, <5, "wh">];

  printa(int_arr);
  return 0;
}
```

test-convert.impv

```
func note[] convertIntToNote(int[] a, int n){
  int i;
  note[] notes;
  note tmp;
  int val;

  notes = [<1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1, "wh">, <1,
"wh">];

  for(i = 0; i < n-1; i = i+1){
    tmp = <a[i]%5, "hf">;
    notes[i] = tmp;
  }

  printNoteArr(notes);

  return notes;
}

func int main() {
  int[] a;
  note[] notes;

  a = [23, 11, 39, 44, 2, 16, 52];
  convertIntToNote(a, 7);

  return 0;
}


test-fib.impv
func int fib(int n) {
    if (n <= 1) {
        return 1;
    }
    return fib(n-1) + fib(n-2);
}

func int main() {
```

```
    printi(fib(0));
    printi(fib(1));
    printi(fib(2));
    printi(fib(3));
    printi(fib(4));
    printi(fib(5));
    return 0;
}

test-for.impv
func int main() {
  int i;
  for (i = 0; i < 5; i = i + 1) {
    printi(i);
  }
  return 0;
}

test-gcd-recursive.impv
func int recursive_gcd(int x, int y) {
    if (x == 0) {
        return y;
    }
    if (y == 0) {
        return x;
    }
    if (x > y) {
        return recursive_gcd(x%y, y);
    } else {
        return recursive_gcd(y%x, x);
    }
}

func int main() {
    printi(recursive_gcd(823, 4115));
    printi(recursive_gcd(98, 56));
    return 0;
}
```

test-gcd.impv

```
func int gcd(int x, int y) {
  while (x != y) {
    if (x > y) {
      x = x - y;
    } else {
      y = y - x;
```

```
    }
  }
  return x;
}

func int main() {
  printi(gcd(823, 4115));
  printi(gcd(98, 56));
  return 0;
}
```

test-if.impv

```
func int main() {
  bool e;
  e = true;
  if (e) {
    prints("true");
  } else {
    prints("false");
  }
}
```

test-note.impv

```
func int main() {
  note n;
  n = <0, "wh">;
  printn(n);
  n = <1, "hf">;
  printn(n);
  n = <2, "qr">;
  printn(n);
  n = <3, "ei">;
  printn(n);
  n = <4, "sx">;
  printn(n);
  return 0;
}
```

test-note2.impv

```
func int main() {
  note n;
  int i;
  i = 1;
  n = <i, "wh">;
```

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

test-note3.impv

```
func int main() {
  note n;
  int[] i;
  i = [1, 2];
  n = <i[1], "wh">;
  printn(n);
  return 0;
}
```

test-printa.impv

```
func int main() {
  int[] int_arr;
  int_arr = [1, 2, 3, 4, 5];
  printa(int_arr);
  return 0;
}
```

test-printbig.impv

```
func int main() {
  printbig(72);
  return 0;
}
```

test-printi.impv

```
func int main() {
  printi(42);
  return 0;
}
```

test-printmidi.impv

```
func int main() {
  note[] arr;
  arr = [<1, "wh">, <2, "wh">, <3, "wh">];
  render(arr, "test.mid", 1, 120);
  printmidi("test.mid");
  return 0;
```

```
}
```

test-printn.impv

```
func int main() {
  note n;
  n = <1, "wh">;
  printn(n);
  return 0;
}
```

test-prints.impv

```
func int main() {
  prints("hello world");
  return 0;
}
```

test-render.impv

```
func int main() {
  note[] arr;
  arr = [<1, "wh">, <2, "wh">, <3, "wh">];
  render(arr, "test.mid", 1, 120);
  return 0;
}
```

test-sort.impv

```
func int[] bubbleSort(int[] a, int n){
  int i;
  int j;
  int tmp;
  printa(a);

  for (i = 0; i < n-1; i = i+1){
    for (j = 0; j < n-i-1; j = j+1){
      if (a[j] > a[j+1]){
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
        printa(a);
      }
    }
  }
  printa(a);
```

```
  return a;
}

func int[] selectionSort(int[] a, int n){
  int i;
  int j;
  int tmp;
  int min_idx;
  printa(a);

  for (i = 0; i < n-1; i = i+1){
    min_idx = i;

    for (j = i+1; j < n; j = j+1){
      if (a[j] < a[min_idx]){
        min_idx = j;
      }
    }

    tmp = a[min_idx];
    a[min_idx] = a[i];
    a[i] = tmp;
    printa(a);
  }
  printa(a);
  return a;
}

func int main() {
  int[] a;

  prints("BUBBLE SORT");
  a = [23, 11, 39, 44, 2, 16, 52];
  bubbleSort(a, 7);

  prints("SELECTION SORT");
  a = [23, 11, 39, 44, 2, 16, 52];
  selectionSort(a, 7);

  return 0;
}
```

test-var-int.impv

```
func int main() {
  int a;
  a = 1 + 2;
```

```
  printi(a);
  return 0;
}

test-while.impv
func int main() {
  int i;
  while (i < 5) {
    printi(i);
    i = i + 1;
  }
  return 0;
}
```

fail-duplicate-function.impv

```
func int add(int x, int y) {
  return x + y;
}

func int add(int x, int y) {
  return x * y;
}

func int main() {
  printi(add(4115, 42));
}
```

fail-inconsistent-arr-type.impv

```
func int main() {
  int[] int_arr;
  int_arr = [1, <1, "wh">, 2];
  return 0;
}
```

fail-inconsistent-return-type.impv

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

fail-rhythm.impv

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

```
  n = <2, "w">;
  return 0;
}
```

fail-tonetype.impv

```
func int main() {
  note n;
  n = <"a", "wh">;
  return 0;
}
```

fail-undeclared-identifier.impv

```
func int main() {
  int i;
  for (i = 0; i < 5; i = i + 1) {
    printi(d);
  }
  return 0;
}
```

fail-undefined-function.impv

```
func int main() {
  printi(add(4115, 42));
}
```

Git Log

```
commit 77f3352a0471c32082292f3803308f1ad8d037e3 (HEAD -> main, origin/main,
origin/HEAD)
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Tue Apr 27 00:59:36 2021 -0400

    Deletes file

commit 577dedd3a2ef34be841d0065b3da37bf5bc38e44
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Tue Apr 27 00:58:20 2021 -0400

    Cleans code

commit d06f7de9e6ec07515ffefd65edf1aedd5115dadf
Author: Emily <el2895@columbia.edu>
```

```
Date:   Mon Apr 26 23:34:24 2021 -0400

    final changes

commit 8ed123e0b657e9aa11ac6bd99553d182a585629d
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 22:56:16 2021 -0400

    append arrays!!! demo programs done!!! wahoo

commit 1d59092731db227b89f9fc93e1d39838e0541ced
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 18:26:29 2021 -0400

    Array append test case

commit 3aea4de091edd4dc75b665ee7fa34d8c96d5e404
Merge: 88992ee f4d8495
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 18:23:14 2021 -0400

    Fixes merge conflicts

commit 88992eef8fd14745c6d62818e75ba1aa4ec54753
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 18:19:31 2021 -0400

    WIP: array append

commit f4d8495479721ba60b01c11527b4987a9df8198d
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 18:06:22 2021 -0400

    modify test files

commit a678dbc8de8f5bb69f39b0cc2b49a286d0719582
Merge: ed66b67 2f55964
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 17:46:13 2021 -0400

Merge branch 'main' of https://github.com/joshchoi4881/improv into main

commit ed66b67a10005ff4642cf4c1a374d49f5fe70fbf
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 17:43:16 2021 -0400

    extra note tests
```

```
commit 479d6624aeb6da2a1daf96cb4a7b54f0f573ccc8
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 17:32:47 2021 -0400

    redoing note struct

commit 2f55964af491539915c30a38c4490417e92d10fa
Author: Josh <joshchoi@Joshs-MBP.lan>
Date:   Sun Apr 25 17:17:39 2021 -0400

    Added more songs

commit 47c33637d81dface16decfe6a5f5869bec9649ac
Author: Josh <joshchoi@Joshs-MBP.lan>
Date:   Sun Apr 25 15:13:31 2021 -0400

    Another One Bites The Dust

commit 5853cbea37dc2cb450af1ee2e97090577b641dc2
Author: Josh <joshchoi@Joshs-MBP.lan>
Date:   Sun Apr 25 14:50:42 2021 -0400

    Old MacDonald

commit 813f0eb77a59addd090e2cf251ec0a88b6951023
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 13:20:46 2021 -0400

    sorting algorithms

commit 82e1f8b861868d385a2ab4e373842ee32e9c8cf6
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 11:47:18 2021 -0400

    build array function

commit 00c953ca1c569e57d55f874f109d0c7e81565ec7
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 11:38:11 2021 -0400

    adding array append

commit a9e24b19db998af4a7a6f376483268316323b990
Merge: 870b5aa aa13560
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 11:10:27 2021 -0400
```

```
    Merge branch 'main' of https://github.com/joshchoi4881/improv into main

commit 870b5aa8c71840ea39d2f2f206d6f334b14e40fb
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 11:10:21 2021 -0400

    ararys were being stored backwards

commit aa135609959ae3816a0fb14c5872ab9e0cc4b639
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 11:08:25 2021 -0400

    Fixes array access and assign

commit 17b3fe2dda389fa9df1dbfce484ba2a5b3689917
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 10:41:57 2021 -0400

    ararys were being stored backwards

commit 1df21a7e1969ab3ff4bf8d2b4a441527a0bce413
Merge: 634629d 6b594ea
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 10:28:57 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv into main

commit 6b594ea9ede79f40c1e72162a793f1ffe6b75b11
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 10:28:33 2021 -0400

    array test cases

commit 634629de655c250a3d29fe03e64c405279a88bd9
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Sun Apr 25 10:25:37 2021 -0400

    arrays in progress

commit 88e78d33b57ee0277dbeb6e91198408d4330ba1a
Author: Emily <el2895@columbia.edu>
Date:   Sun Apr 25 09:29:51 2021 -0400

    successfully linked printn function to print notes

commit 6ff53933907c6c895ea12b33aae4ac29233af207
```

```
Author: Emily <el2895@columbia.edu>
Date:   Sat Apr 24 17:57:11 2021 -0400

    c functions to create midi file

commit 28313c9bc6ca19951350bc763c64fcab4273e544
Merge: 3b657e2 22276c7
Author: natalia <nataliadorogi12@gmail.com>
Date:   Sat Apr 24 15:21:14 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv

    .

commit 3b657e2b5dd374d78824a26363b2452c39e153ce
Author: natalia <nataliadorogi12@gmail.com>
Date:   Sat Apr 24 15:18:17 2021 -0400

    updated tests with parans and added more error tests

commit 22276c7ef23fe0e6fdd831cadb9392c572d3145c
Merge: 5474281 4fab2c1
Author: Emily <el2895@columbia.edu>
Date:   Sat Apr 24 13:58:31 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv into main

commit 4fab2c1012be9a4a3caf2c13a0ce6187c791e696
Author: natalia <nataliadorogi12@gmail.com>
Date:   Sat Apr 24 13:58:16 2021 -0400

    updated for render function and notes

commit 54742818a465115db6493e7119ba735e4d5d401c
Author: Emily <el2895@columbia.edu>
Date:   Sat Apr 24 13:56:48 2021 -0400

    wrote print function for notes

commit b1aeec46b427e69bddead0d8bfaa2f00bedd84a3
Author: Emily <el2895@columbia.edu>
Date:   Sat Apr 24 13:15:00 2021 -0400

    added key definitions

commit 9991a16836004db2156bdb931172de8b740d26ce
Author: Emily <el2895@columbia.edu>
```

```
Date:   Sat Apr 24 12:57:10 2021 -0400

    testing how to link c files to our program

commit 8b2fdc1cdc01d6d2311df5100f55ee03d884afc9
Author: Josh <joshchoi@Joshs-MBP.lan>
Date:   Fri Apr 23 18:26:28 2021 -0400

    Added major and minor pentatonic scales

commit 3506a960a9f01098a0664f9c0542b1e0919fcfc1
Author: Emily <el2895@columbia.edu>
Date:   Fri Apr 23 18:00:18 2021 -0400

    adding c library to write midifiles

commit a1779a350ea6f40f410676fb853b981ba1301f3b
Merge: c8de915 d83fc91
Author: Emily <el2895@columbia.edu>
Date:   Fri Apr 23 17:47:03 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv into main

commit d83fc91b31e423dac3c8ac2b2faf5d6ea94fb978
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Fri Apr 23 17:32:37 2021 -0400

    Working on arrays, wip does not work

commit c8de915940bd88b0a8e3c70d6c031429f2daca13
Author: Emily <el2895@columbia.edu>
Date:   Fri Apr 23 17:21:18 2021 -0400

    change valid tone values

commit 6e54fc92e683dfe9235ba36f3de9ad3fb490ddc3
Merge: f5a50eb c74635b
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Fri Apr 23 15:02:41 2021 -0400

    Who knows

commit f5a50ebd780cd73fe58c48c8f6012071353b0b10
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Fri Apr 23 15:01:45 2021 -0400

    Who knows
```

```
commit c74635bee72c16744249edde9313e875c20c5dbe
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 22:44:01 2021 -0400

    array tests

commit 46171e1f4ca01bbff2bceecbab1aac54e846c932
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 22:32:01 2021 -0400
fix gcd error

commit da0fad589ce5c62d72b4ed1ddab9e6575f3d241b
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 22:18:25 2021 -0400

    fix gitignore

commit 85814a20aa4974ef8daaf8ad298197d9cf9f02f3
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 22:10:55 2021 -0400

    added gitignore file

commit 53f4ce12ddc1c8bdf6a31a5eb9f1d5652b163906
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 22:06:38 2021 -0400

    sample demo program

commit 99db098f947aab33a9722fb427d58e74804dfc78
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 21:54:17 2021 -0400

    added more tests cases and script to run all tests

commit ae520f30d88bdbb43f0e59e308403332f4bcb87f
Author: natalia <nataliadorogi12@gmail.com>
Date:   Thu Apr 22 20:59:02 2021 -0400

    changed for loop functionality

commit 2bdf908eb482e60f9bffd15ef8d64f880adcf889
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Thu Apr 22 20:27:30 2021 -0400

    Adds arrays to codegen
```

```
commit 6724374bd94bdafb0a1eafe5215872afe05012cd
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 18:52:28 2021 -0400

    fix string equality check

commit 535e9b2022905666c7b4d52657a7f7d8d0ecf8d1
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Thu Apr 22 18:46:12 2021 -0400

    WIP: Adding notes & arrays to codegen

commit b3f385e933a6f145c96dc6baa36cc0f0383af74b
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 16:42:01 2021 -0400

    updated note, tone, rhythm

commit fd492f392c70decac522720f52f2622b9d256d10
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 16:24:09 2021 -0400

    added tone, rhythm, note, arrays in semant

commit 5106d32af35c94823d63268cd1fed4f5328a81ce
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 22 13:15:22 2021 -0400

    semantic checking for note and rhythm?

commit b0ad0a910878238104626ff8f5c3547dd4599242 (origin/emily)
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Apr 21 22:12:33 2021 -0400

    Fixes lookup issue

commit 3f3694c171b824b7e30229f0fc451b4f5946fc7c (origin/c-function)
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 21:13:51 2021 -0400

    more tests

commit 6b1a21a7f3710d67547e61b5cf3b2f8d4becf3e5
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 19:03:45 2021 -0400
```

```
    wrote some tests

commit 36c7dfca1ced81fef865c469548d11ac3086626d
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Apr 21 18:56:53 2021 -0400

    Fixes Hello World

commit fd509d65344202cde6a088bf5aa68a5a52d70fca (origin/update-language)
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 15:33:46 2021 -0400

    delete files

commit 71031e2e8703ce5440ed3e80c308fb233b73647a
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 15:33:24 2021 -0400

    minor updates
commit 16beb565d7d7cc150e55342c0004f97792e6eda6
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 14:52:16 2021 -0400

    update semant

commit 353e0a97d5aff060cdf5d64e2acc6f79b10f0235
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 14:47:34 2021 -0400

    organize files

commit 736550c6615abfd69dbe969e2adf406ade58ff68
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 14:46:19 2021 -0400

    .

commit 1e440a3001f2a64513919346f7d7434117ca2fa6
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 14:45:43 2021 -0400

    update language changes in ast, sast

commit e24a2429158724165f3255a17b87237da9cfbb3e
Author: Emily <el2895@columbia.edu>
Date:   Wed Apr 21 14:42:03 2021 -0400
```

```
    remove maps, decorators, add mod operator

commit 5722b26eafb13ccee942290baef869b49ce63471
Merge: d79f47b d648f5c
Author: Emily Li <50225176+emilyili@users.noreply.github.com>
Date:   Thu Apr 1 21:24:14 2021 -0400

    Merge pull request #4 from joshchoi4881/semantic-checking

    re-organizing files

commit d79f47b8d10a6b2f92c35e1030b3e3e4d7f8c2db
Author: nataliadorogi <nataliadorogi12@gmail.com>
Date:   Thu Apr 1 21:22:12 2021 -0400

    Update README.md

commit d648f5c2b4250b18779cd24e65c04f952c5387c3 (origin/semantic-checking)
Author: Emily <el2895@columbia.edu>
Date:   Thu Apr 1 21:21:30 2021 -0400

    re-organizing files

commit 166946ece4ad534996d8506142085a6bd5b17d9b
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Mar 24 21:45:48 2021 -0400

    Fixes string type in codegen

commit 6c722ad3acf420a141d99699b32cea86d9e93b1a
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Mar 24 19:49:23 2021 -0400

    Built improv.native

commit 0d5f80c8001d159f65110cfc224db6db5cdb1747
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Mar 24 19:14:44 2021 -0400

    Fixes hello world bugs

commit 09dc3812c546e97dfc14d07014709f0828521d35
Merge: 9be6717 dc5360c
Author: Emily <el2895@columbia.edu>
Date:   Wed Mar 24 18:12:45 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv into main
```

```
commit 9be6717b616d8e14c724fa32b76315e96e2a22c0
Author: Emily <el2895@columbia.edu>
Date:   Wed Mar 24 18:06:16 2021 -0400

    update sast

commit dc5360c998989adbab4c8cb4163502d9e44dbd67
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Wed Mar 24 17:20:16 2021 -0400

    Still fixing hello world build

commit 52f55d7be8b730fa00931c7e4644cdb30d5f39ca
Merge: 98e0b34 852f7c2
Author: natalia <nataliadorogi12@gmail.com>
Date:   Wed Mar 24 11:30:36 2021 -0400

    Merge branch 'main'  f https://github.c m/joshchoi4881/improv
    for building

commit 98e0b3431e080e069bd84cb0b2af78b0e54ddd06
Merge: a5d0285 0c1c597
Author: natalia <nataliadorogi12@gmail.com>
Date:   Wed Mar 24 11:30:30 2021 -0400

    for building

commit 852f7c282db63f6f72ef3aa9c0476917e0c617c5
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Tue Mar 23 21:35:13 2021 -0400

    Preliminary codegen

commit a11a8ef49eccb2666bdceabbbae45a78f2727a45
Author: alice-zhang <32142194+alice-zhang@users.noreply.github.com>
Date:   Tue Mar 23 20:17:25 2021 -0400

    Update compiler.ml

commit 8c7eb44c7816d9670f96ac9de7979a112f656790
Author: alice-zhang <32142194+alice-zhang@users.noreply.github.com>
Date:   Tue Mar 23 20:07:19 2021 -0400

    Update compiler.ml

commit a5d028590a5145aef47f8547d12169e90b121954
```

```
Author: natalia <nataliadorogi12@gmail.com>
Date:   Tue Mar 23 19:38:05 2021 -0400

    created makefile :-)

commit 0c1c597d37e65a751abe1b1ff4f579acfe626562
Author: Emily <el2895@columbia.edu>
Date:   Tue Mar 23 18:05:00 2021 -0400

    static semantic checking

commit c88d122d0458d72f6d34a5f40287fd758943d3fe
Author: natalia <nataliadorogi12@gmail.com>
Date:   Tue Mar 23 17:19:24 2021 -0400

    semantic check v1

commit 50b5a1665af7543c3f89246b056e7b4912f6eb41
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Tue Mar 23 15:52:02 2021 -0400

    Edits ast

commit 20d31712e19820c5117d67451bbe2fc79fbaa626
Merge: c0722f1 da757a7
Author: Josh <joshchoi@Joshs-MBP.home>
Date:   Fri Mar 19 15:36:38 2021 -0400

    Merge branch 'main' of https://github.com/joshchoi4881/improv into main
    Merge

commit c0722f159312894dce07d05f8f122e05413d8e6f
Author: Josh <joshchoi@Joshs-MBP.home>
Date:   Fri Mar 19 15:36:33 2021 -0400

    AST file

commit da757a71a49e16c907afb6babaddff769f159826
Author: alice-zhang <ayz2105@columbia.edu>
Date:   Fri Mar 19 11:36:39 2021 -0400

    Adds preliminary top-level compiler

commit dae03c5a4f228b46d0a4f6f722999626a191729d
Merge: 940af84 1bcc767
Author: Josh <joshchoi@Joshs-MBP.home>
Date:   Thu Mar 18 21:59:50 2021 -0400
```

```
    Merge branch 'main' of https://github.com/joshchoi4881/improv into main
    Merge

commit 940af84ab0f3782f8d5b392227b7753d6f4e1efb
Author: Josh <joshchoi@Joshs-MBP.home>
Date:   Thu Mar 18 21:59:33 2021 -0400

    Progress on ast file

commit 1bcc7676e10a451830f136f1b5536eb24acb6eb1
Author: Emily <el2895@columbia.edu>
Date:   Wed Mar 17 17:13:12 2021 -0400

    fixed shift/reduce errors

commit c8814b296445970ee8b25ccaaebd34a93f1b9687
Author: Emily <el2895@columbia.edu>
Date:   Mon Mar 15 22:49:43 2021 -0400

    fixed 9 shift/reduce conflicts, one to go

commit e4df08d19bd7f8d904508dad0a8364235be3f5e4
Author: Emily <el2895@columbia.edu>
Date:   Mon Mar 15 13:59:22 2021 -0400

    finished parser,  working on shift/reduce errors

commit bdc738636ab7df5396b49aaa0a3f783b3b5de98b
Author: Emily <el2895@columbia.edu>
Date:   Wed Feb 24 22:16:45 2021 -0500

    parser draft

commit abe75758be4471c493c42aaa9b2c26b0ea2d4f91
Author: Emily <el2895@columbia.edu>
Date:   Wed Feb 24 18:47:11 2021 -0500

    minor fix

commit daf28a57ff62e5c58079a5d7ee14afc47709d406
Author: Emily <el2895@columbia.edu>
Date:   Mon Feb 22 22:40:30 2021 -0500

    added decorator

commit 9989112b8f8f5a9499d5c6ade0c4b33dac4dcd46
```

```
Author: Emily <el2895@columbia.edu>
Date:   Mon Feb 22 22:27:28 2021 -0500

    fixed compilation errors

commit 28741c5d40dd32489dc0a22aa5671a0c960068a9
Author: Emily <el2895@columbia.edu>
Date:   Mon Feb 22 22:17:10 2021 -0500

    scanner

commit 368c6744438510f40f8894eb639d4dad32a89a14
Author: Emily <el2895@columbia.edu>
Date:   Mon Feb 22 21:24:11 2021 -0500

    starting to write scanner

commit c08aa8ebb2f6fa65b26d50139115395fd7fc320a
Author: Josh Choi <jc4881@columbia.edu>
Date:   Mon Feb 22 20:30:07 2021 -0500

    Initial commit
```