Improv 🎵

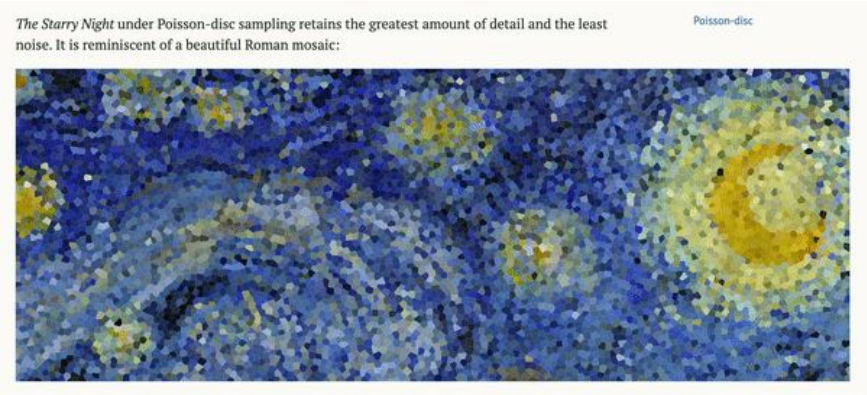
Alice (Tester), Emily (Architect), Josh (Language), Natalia (Manager)

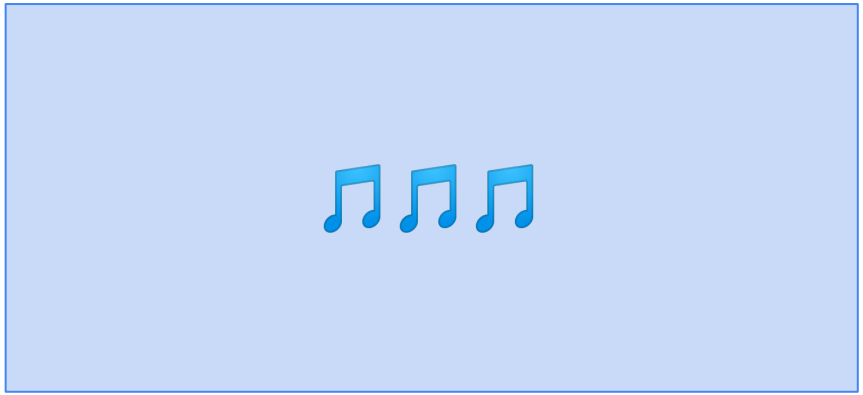
Overview

1. Background and motivation
2. Improv language
    a. Syntax
    b. Structure
    c. Architecture
    d. Features
    e. Making Music
3. Future works
4. Demonstration

Background & motivation 🎵

Fighting for music hum representation of Algorithms





Get it? Cuz this is like art for art hum?

Source: https://bost.ocks.org/mike/algorithms/

Music and Math ➕

Algorithms can get us to an answer, but in different times and different ways. Algorithms can be visualized - BUT we wanted to our ears to understand algorithms.

What differentiates what an algorithm sounds like? Runtime, efficiency, general approach

Source: wxqr.org

Improvise music, manipulate algorithms & create music

```
func note[] gcd(int x, int y) {
     note[] result;                                    // initialize variables
     int c;
     result = [];
     while (x > 0) {                                   // GCD while loop
        result = append(result, [<x%5, wh>];           // add note to array
        result = append(result, [<y%5, wh>];           // add note to array
        a = x % y;                                     // mod
        x = y;                                         // reassignment
        y = c;                                         // reassignment
    }
     result = append(result, [(y%6), wh)];             // add note to array
     printNoteArr(result);                             // print array
     return result;                                    // return
}
func int main() {
     note[] result;
     result = gcd(36, 125);                            // call GCD
     render(result, "gcd.mid", 1, 96);                 // create music file
}
                                                       // jam to music output
```

Implementation 🎵

Syntax

- Functions are defined as **`int main() {}`**
- We use curly braces **`{}`** for scoping
- End lines with **;** ⟶ white space is ignored
- And of course there are musical terms...
- Variables are declared with their **type**!
  - Standard data types include `int, bool, string, void`
  - Improv data types include `note, tone, rhythm`
- Print is specific based on types i.e. `printn()` -> prints note

<1, “wh”>

Program Structure

- Statically scoped
- Declarations happen before calls (for functions, variables, and arrays)
- Strong, static typing
- Control flow is pretty standard

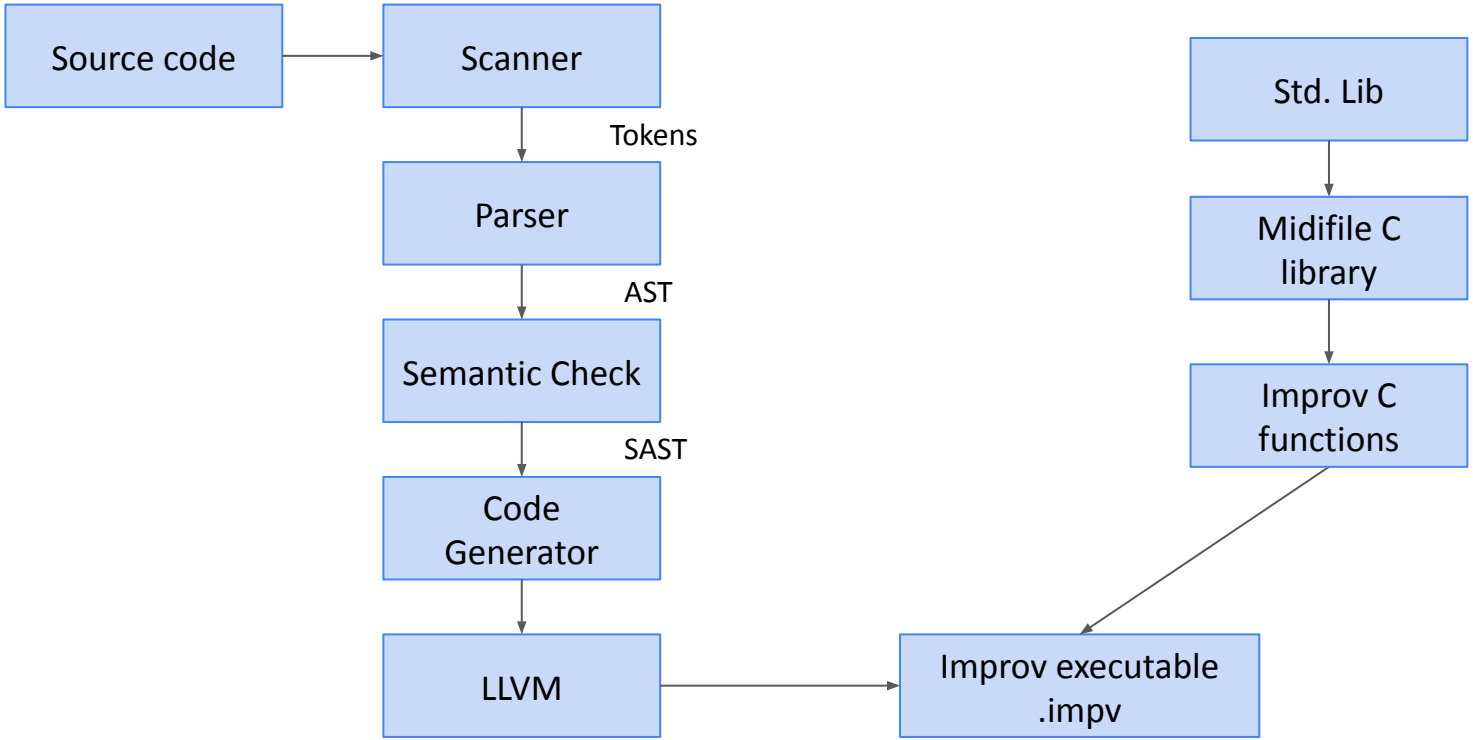
```
func int main() {
  int i;
  for (i = 0; i < 5; i = i + 1) {
    printi(i);
  }
  return 0;
}
```

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

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

Built in function: `render` to create a music file from numbers, notes, and a key!

Compiler Architecture



Arrays

- Arrays can be formed of types `int`, `string`, and `note`
  - string[] t;
  - t = [“alice”, “emily”, “josh”, “natalia”]
  - + assign, access, append!
- Implemented using a fat pointer
- Stored on the heap

```
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;
}
```

Notes tone ‘n rhythm

`tone`: pitch of note

Represented by ints `0-5` and map to differenerent tonalities on the pentatonic scale specified by a key -> taken care of in semant.ml

`rhythm`: flow of sound

Represented by two char strings named intuitively; i.e. `wh` = whole note, `hf` = half note, `qn` = quarter note, `ei` = eighth note, `sx` = sixth note

`note`: struct data type encompassing the tone and rhythm

`<1, “wh”>` in CMAJ represents a whole note in C

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

render

- Built in function that is used to create music files
- Calls on [Steve Goodwin's pure C library]
- Linked similarly to how printbig is

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

```c
/* 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);
}
```

Testing

- Many many tests… especially for MIDI files
- Iterative testing process

```
root@7323e6714e48:/home/improv# ./testAll.sh
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
```

Next on the roadmap 🎵

Future improvements

Inclusion of different styles like blues, which adds one note to the pentatonic scale

Inclusion of pre-written riffs to include in improvisation

Ability to choose different sounds and instruments

Demo 🎵

Bubble sort + Selection sort

McDonald’s… yum

Thank you 😁