October 16 PLT Meeting

Progress Report

- MIDI seems do-able but need to make some changes
- Make . a postfix operator
  - Can have multiple dots
- Add %+ and %- for PC addition and subtraction (mod 12)
- Formatting in code: 4 spaces

Future Work

- Slices (*easy*, medium, **hard**)
  1. Lianne
  2. Kuangya
  3. Lindsay
  4. Van
  5. Richard

  - *(1) Literals*
  - (1) Main/Print/Random
  - (2) MIDI generation
    - [ ], Note, Chord, System
    - What needs to be changed?
    - What does smurfy-code look like
    - What does the MIDI generator need to see
  - *(3) Operators (non-music)*
  - *(3) Notes/Beats and operators*
  - **(4) Functions**
  - **(5) Pattern Matching**
  - **Guards** (might die)
  - Bindings
    - Function declaration
    - Definitions
    - Type Specifications
    - Let
  - *Conditionals (if-then-else)*
  - *Lists (including Chord and System)*

What is a program in SMURF?

- List of declarations
- main is a keyword that evaluates an expression that is something musical
  - Something musical: do we want to restrict to just a list of systems? No
    - Empty List (so that we can print something)
      - main = print (2)
        [ ]
      - main = print (getNote 0 1 2 .)
    - Note

        - Chord
        - System
        - [System]
        - Return a compile time error if not one of the above things
- print is special function that prints whats in the parentheses that then evaluates to that expression (just a special kind of expression)
    - (print x) = x
    - print goes to standard output (new line between each print statement)
- Input to SMURF? No
- Output from SMURF?
    - print goes to standard output
    - MIDI: 2 options
        - CVS file in specified MIDI format

          Notes:

(time)	(note)
0	0
2	0
4	0
8	0

          Chord:

(time)	(note)
0	0
0	3
0	5
0	7
        - Develop our own “smurfy-code” format (ASCII text) that gets passed to MIDI converter (in java) that outputs a .midifile

LRM

- Type synonyms for readability
    - [Note] = Chord
    - [Chord] = System
- Primitive types
    - Int, Bool, Beat
- Get rid of Tuple
- Derived Types
    - What does base one represent? Easiest to start with Treble 1 (look at proposal)
- Expressions
    - Additions: specify all expressions have a type
    - **Description of Precedence: TODO**
    - Add examples for each of the expressions
    - Higher order functions?
        - Partial Application: If specify function with certain number of arguments, you can call that function with less than that number of functions

          f :: Int → (Int → Int)

          (+) 3 3 → 6

(+) 3 → function waiting for a second argument
Map ((+) 3) [1, 2, 3] → [4, 5, 6]

- Good: works well with compiler
- Bad: might be hard to do (“tricky but doable”)

- Currying: Pass functions to other functions as arguments

- Let: specify new line syntax
- Pattern Matching and Guards: leave for later