October 23 PLT Meeting

**** No meeting next week**
Hard deadline on Nov 6th meeting: Completed Parser with unit tests (no conflicts. etc)

- 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) Function Application**
  - **(5) Pattern Matching**
  - **Guards** (might die)
  - (5) Bindings
    - Function declaration
    - Definitions
    - Type Specifications
    - Let
    - Polymorphism
  - *(2) Conditionals (if-then-else)*
  - *(2) List expressions (including Chord and System)*

Progress Report

- Working on shift/reduce conflicts on Beat and Note in Parser
- MIDI - "smurfy-code" should look like csv
  - 36 - 83 values in MIDI Notes, 0 for rest
  - Sound = Velocity 90, Rest = Velocity 0

  -
    ```
    0  36   90
    0  48   90
    0  60   90
    1   0    0
    4  50   90
    ```
- LRM
  - Pattern Matching more simple than expected
  - We do want Guards
    - For f < 10

```
f  x
    | x < 10 = <do something>
    | x > 10 = <do something else>
    | otherwise = <do a third thing>
```

- Declarations
  - Add global variables (to declare PC row at beginning)
  - Need a type signature for a function declaration
  - Do not need a type signature for a definition (type inference for variables should be pretty simple) but can have type signatures

THE FUTURE

- How to organize these steps? Modularize steps
  - Semantic Analysis
  - Translation to SMURFy code
  - Output translated SMURFy code to MIDI
- New Architecture:
  SMURF → Scanner/Parser → AST → Semantic Analyzer → SAST* → Translator → SMURFy code → MIDI Converter → MIDI
  *Semantic AST
- Technique for testing your part when someone hasn't done theirs: Create a "Dummy" that outputs what you need to test (ex. Function that always returns a bool when you need to test how your part works with bools)
- Use future meetings as coding sessions to organize parts of the code that doesn't fall into anyone's particular section
- Testing
  - Write tests of parser and scanner
    - One positive for each part
    - And negative cases: should fail