Curtis Henkel
Matthew Duane
Chatura Atapattu
Kevin Ramkishun

TONEDEF

Motivation

- Create a language for the “Musical Computer Scientist"
- Bring the semantics of musical composition to a programming language
- Flexible enough for a variety of uses
  - Simple music creation
  - Algorithmic music creation

Language Overview

- Imperative programming language
- Statically scoped, weakly-typed
- Types are immutable
- No pointers
- No polymorphism of function names
- /* Comments */
- Source file is piped in and optional output file can be piped out.

Types

int	digits only
boolean	true or false
string	chars inside double quotes
pitch	$[A-G][b#]?[0-9]?
sequence	comma-separated list of ints inside [ ]
beat	rational numbers
note	pitch + beat
chord	set of notes
phrase	ordered list of chords
rhythm	1,0,-,_ within ' '
void	void

Operators

- Arithmetic: `+ - * / // %`
- Comparators: `< > <= >= == !=`
- Unary: `- !`
- Note/Chord Manipulation: `^ ^^ : :: +`
- Phrase Manipulation: `<< >> ** @@`
- Assignment: `=`

Function Declarations

```
type function func_name (type name list) {
  body
}
```

- Program is a list of function declarations
- Applicative order
- Pass-by-value parameters
- Special functions
  - `main`
  - `print`
  - `play`

Syntax Overview

expression := literal | unop expr
| expr binop expr
| name (expr list) | name = expr
| [expr list] | name

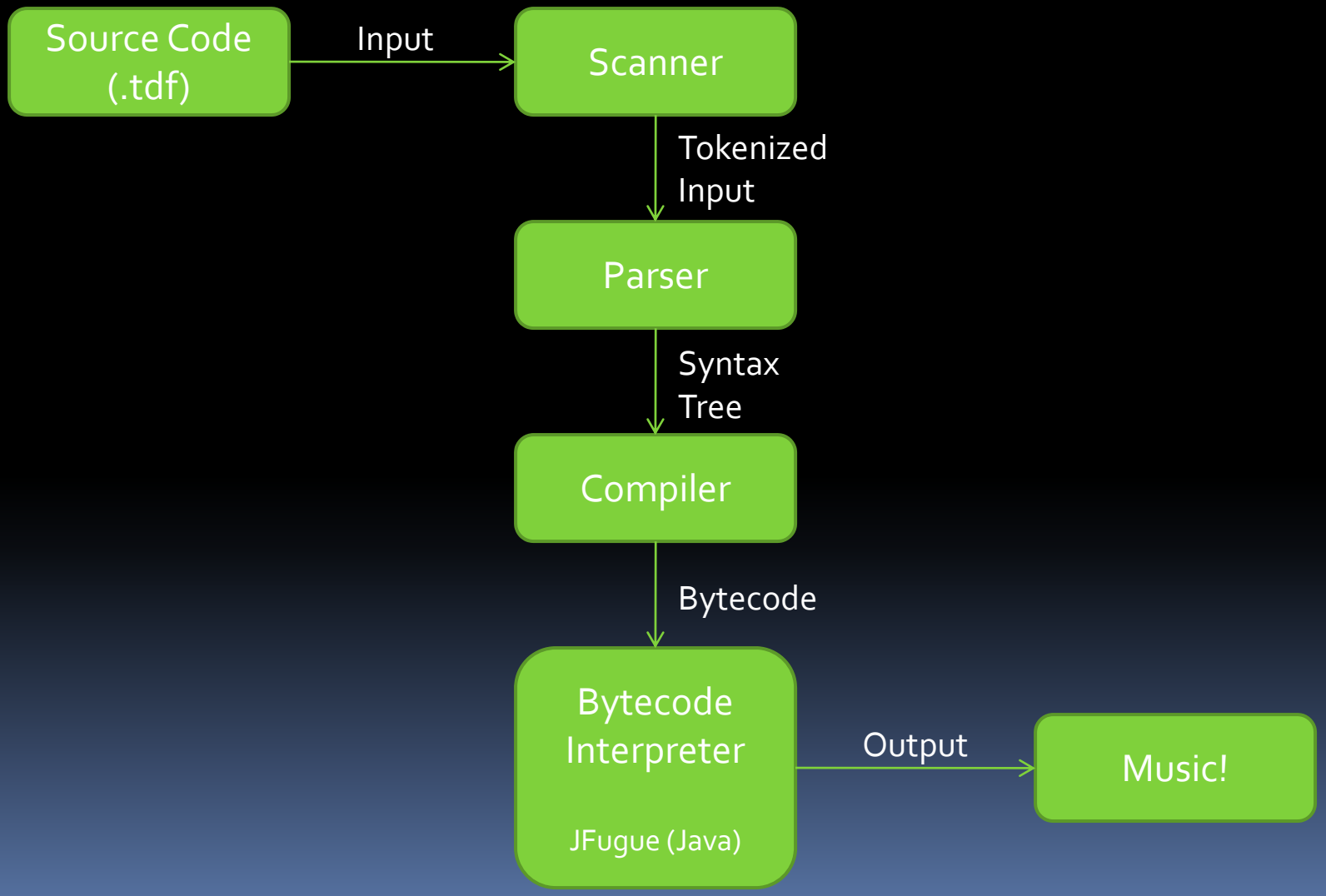
statement := expr; | if (expr) stmt [else stmt]
| for (expr; expr; expr) stmt
| while (expr) stmt
| foreach(type name in expr) stmt
| return expr;
| {stmt list }
| type name [= expr];

Example Program - Phrase Creation

```
void function main() {
    note n = $C5 : 1//4;
    sequence s = [];
    for (int i = 0; i<10; i = i+1){
        s = s + [x]; /* Creates a sequence
    }
    phrase p = n << reverse_sequence(s); /* Creates phrase using note
                                            and sequence */
    play(p);
}

sequence function reverse_sequence (sequence in_seq) {
    sequence s = [];
    foreach (int x in in_seq) {
        s = [x] + s;
    }
    return s;
}
```

Language Implementation



Sample Bytecode

```
void function main() {
   foreach (int i in [4,5,6]) {
       print (i + "\n");
   }
}
```

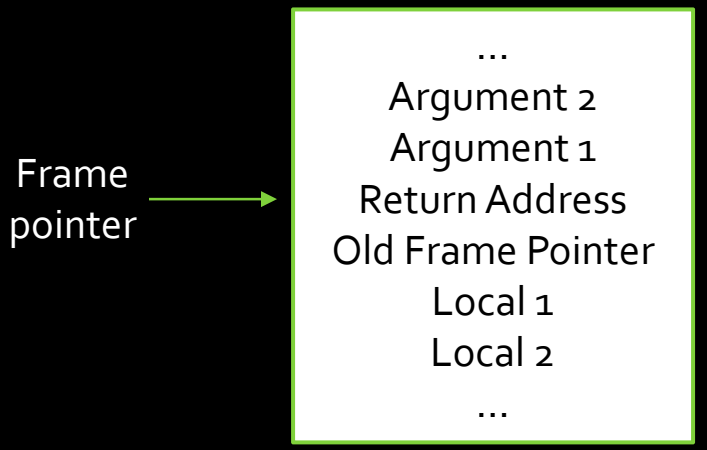
Output:

```
4
5
6
```

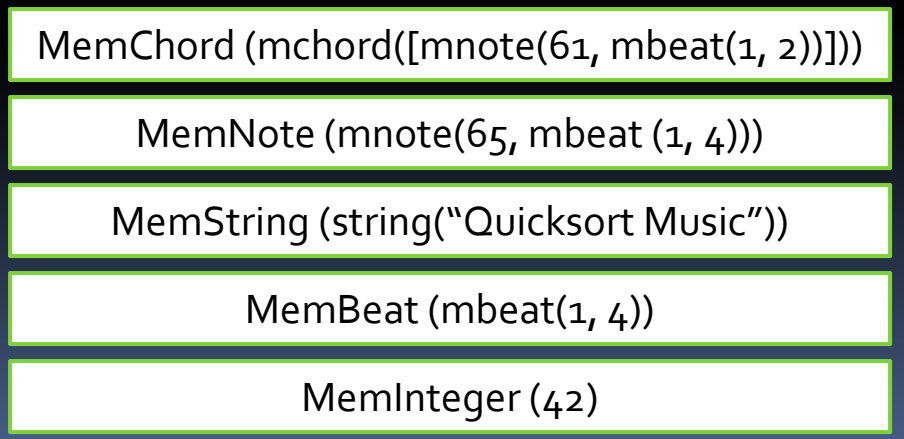
```
0 Jmp 2
1 Hlt
2 Entry 1
3 PushEmpty sequence
4 PushInt 6
5 Combine
6 PushInt 5
7 Combine
8 PushInt 4
9 Combine
10 Bra 10
11 Decompose
12 Store 1
13 Pop
14 Load 1
15 ConvertType int ->string
16 PushString \n
17 Concat
18 Jmp -1
19 Pop
20 IsEmpty
21 Beq -10
22 Pop
23 PushInt 0
24 Ret 0
```

Stack

- type mbeat = int * int (* numerator, denominator *)
- type mnote = int * mbeat (* pitch, beat *)
- type mchord = mnote list (* list of notes *)
- type mphrase = mchord list (* list of chords *)
- type msequence = int list



- MemInteger of int
- MemString of string
- MemPitch of int
- MemBool of bool
- MemBeat of mbeat
- MemNote of mnote
- MemChord of mchord
- MemPhrase of mphrase
- MemSequence of msequence
- MemRhythm of string



How We Collaborated

- Source Code Control - Subversion
  - Hosting by Assembla.com
  - Stress frequent commits
  - Automatic E-Mail on new commits to notify team members
- Google Documents
  - Collaborative editing on all documents and presentations
  - Proposal, Task List, LRM, Presentation, Final Report
- Instant Messaging, E-Mail conversations
- Impromptu meetings (after class) as required

Lessons Learned

- KISS - Keep it Simple Stupid (cliché, we know)
  - Everything is just an integer or a string
- If you don't have an easy solution, just add another layer of indirection
  - Orthodox stack -> Object based stack

Summary & Conclusion

- Successfully implemented Tonedef per the Language Reference Manual
- Future work
  - Overlapping phrases
  - User interaction in program
  - Command arguments
  - Different instruments, tempo and time signature