Tonedef: A Language for Manipulating Music

Kevin Ramkishun Chatura Atapattu
Curtis Henkel Matthew Duane

December 22, 2010

Contents

Chapter 1

Introduction

1.1 Tonedef White paper

Tonedef is an imperative programming language designed to represent and manipulate the components of musical score. Its basic data types are chosen from the lexicon of music and its operators chosen to provide basic transformations on these types. In addition, it provides many of the features common to other popular languages, such as control-flow statements, limited automatic datatype conversions, and user-defined functions. Tonedef aims to provide a platform for constructing programs using abstractions that are familiar and useful to developers with at least a basic knowledge of music theory.

The following report is intended for programmers of the Tonedef language. It explains the available components of the language and how they can be combined to build a Tonedef program, in addition to chronicling the evolution of Tonedef from proposal to a fully-realized "little language".

1.1.1 A Quick Summary

Tonedef is

- Statically Scoped
- Weakly Typed
- Imperative
- Applicative Order
- Bytecode compiled & Interpreted
- Fun!

1.2 A Note on Navigating This Document

Being that this is a rather large document, there are a few methods available for navigation. The table of contents is click-able and will take you directly to a section. The side bar also contains bookmarks that can aid in navigation. Lastly, any reference to a Section or Figure in the book (ex. the "A.1" in "See Section A.1") is also click-able.

Chapter 2

Language Tutorial

Tonedef features a nomenclature and structure that shares many of the conventions common to modern programming languages, making development intuitive to users accustomed to programming in other imperative languages such as C. This section outlines how to utilize Tonedef in the creation and manipulation of music through the use of a sample program that highlights many of the key features found in the language.

First a sample program is presented that contains two functions that are used to build up a sequence of notes. This sample program is then analyzed in depth in Section 2.3.

2.1 Sample Program

```
/* Recursive function for computing the Greatest Common Denominator between two ints */
int function gcd(int a, int b) {
        if (a % b == 0)
                return b;
        else
                return gcd (b, (a % b));
}

/* Iterative function for computing fibonacci series for int n, returning a sequence */
sequence function fib(int n) {
        int a = 0;
        int b = 1;
        int i;
        int sum;

        sequence s = [];

        for(i = 0; i < n; i = i + 1)
        {
                s = s + [a];

                sum = a + b;
                a = b;
                b = sum;
        }

        /* return completed sequence */
        return s;
}

/* the main function that is called initially whenever the program is executed */
void function main(){
        print (``A sample program for Tonedef: \n'');
        note n = $C4 : 1 // 4;
        sequence s = fib(10);
        s = s + [gcd(5552368, 42)];
        phrase p = n << s;
        play(p);
        print(``Phrase = '');
        print(p);
        print("\n");
        return;
}
```

2.2 Executing the Sample Program

- If not done so already, compile the Tonedef executable by typing "`make`" in the source directory.
- Run the tutorial file by typing "`./tonedef < tutorial.tdf`". If desired, the output can be rerouted/"piped" from the screen to a file by adding "`> outputfilename`" to the end of the line above.
- If you are using Linux and cannot run the Tonedef or cannot hear any audio, uncomment line 104 in `execute.ml`. This changes the command string to include "Pulse Audio" which is required on some systems.

2.3 Walking Through the Sample Program

This section goes into greater detail about each function as well as highlight some of the nuances to the language. All functions in Tonedef must be declared in the following manner:

return_type `function` function_name (parameter_list)

- return_type: The Tonedef data type that will returned from the function. Note that return_type may be the special type `void` for functions that return no value.
- `function`: A keyword in Tonedef, it must be included in every function declaration between the return_type and the function_name.
- function_name: A name for the function that will be used to reference it throughout the program. The function_name must be a valid identifier as defined in the Language Manual portion of this report.
- parameter_list: A list of zero or more comma-separated parameter declarations where each declaration is of the form `type identifier`. `type` must be a valid Tonedef data type and `identifier` must be a valid identifier.

Additional information relating to specific elements of the language can be found in the Language Manual portion of this report.

2.3.1 `gcd` function

The function `gcd` is a recursive function that accepts two arguments `a` and `b` of type `int` and returns an `int` which is the greatest common denominator between the two arguments. Line 3 features an `if-else` statement that uses the `%` operator to determine the remainder between `a` and `b` and then compares that value with 0 using the equality operator `==`, which implicitly returns a `bool` value of `true` if the remainder is equal to 0 and `false` if not. If the two values are equal, `gcd` returns the value of `b` with the keyword `return`; if not, `gcd` is recursively called with the values of `b` and `a % b` until `a % b` is equal to 0.

2.3.2 `fib` function

The function `fib` is a function that accepts one argument `n` of type `int` and returns a `sequence` composed of the the Fibonacci series for `n`. Lines 11 thru 16 declare the five variables to be used in the function: variables `a`, `b`, `i`, and `sum` of type `int` and `s` of type `sequence`. Variables `a`, `b`, and `s` are also initialized with their respective values at declaration while `i` and `sum` are not; this is perfectly acceptable, as Tonedef supports declarations of variables with or without initial values.

After the variable declarations, a `for` loop is defined in lines 18 thru 25 that generates the Fibonacci series for the variable `n` using the Fibonacci algorithm. The first expression in the `for` loop set `i` to 0; it will serve as the counter for the loop. The next expression, `i < n`, serves as the predicate for the loop, meaning it will continue to iterate as long as the value of `i` is less than the value of `n`. The final expression, `i = i+ 1`, is called implicitly at the end of each iteration of the `for` loop, in this instance increasing the value of `i` by one.

During each iteration of the `for` loop, the value of `a` is automatically cast from an `int` to a `sequence` of one value by placing it within the brackets [ ]. This sequence is then appended to `s` by the + operator, with the resulting new `sequence` stored back to the variable `s`. The next value of `a` is then computed in the loop. Once the loop completes, `s` is returned from `fib` and the function terminates.

2.3.3 `main` function

The program begins in `main`, where the Tonedef system function `print` is called with the `string` parameter so that it may be displayed on the screen (or piped file as described in Section 2.2). Line 34 declares the variable `n` of type `note` and initializes it as a C4 quarter-note, and Line 35 defines `s` of type `sequence`. `s` is initialized by calling the function `fib`, which returns a `sequence` containing the Fibonacci series for the argument 10. In the next line, the function `gcd` is called with `int` arguments 5552368 and 42 and the return `int` value is then cast to a `sequence` of one value by placing the `gcd` function call within the brackets [ ].

This `sequence` is then appended to `s` by the + operator, with the resulting new `sequence` stored back to the variable `s`. Line 36 creates a new variable `p` of type `phrase` and stores as its value the resulting `phrase` created by the application of `sequence s` to the previously-defined `note n` using the sequence application operator <<. Line 37 calls Tonedef system function `play` with the newly-created `phrase p` as its argument, which results in the tune being played through the user's sound card/speakers. The next couple of lines use the `print` function to display both a `string` and the completed phrase `p`. Line 42 returns the implicit value `void` to the `main` function, which in turn terminates the program and returns control back to the operating system.

Chapter 3

Language Reference Manual

3.1 Lexical Convention

Tonedef parses characters from source files into five types of tokens: identifiers, operators, keywords, punctuators and literals. Blanks, tabs, newlines, and comments (collectively, “Whitespace character(s)”) are generally ignored except insofar as they are used to separate tokens. At least one Whitespace character is required to separate identifiers or literals. Where applicable, appropriate usage of Whitespace character(s) shall be included in the definition for the given lexical token.

3.1.1 Comments

Comments begin with the first character sequence `/*` and end with the first character sequence `*/` that is encountered.

Example:

```
/* This
is a
comment */
int i = 1 /* So is this.  */
```

3.1.2 Identifiers

An identifier is a sequence of alphanumeric characters, with the underscore character “_” included in the alphabet. Identifiers must begin with an alphabetic character, and may be followed by an optional series of one or more alphanumeric characters. Identifiers are case-sensitive, thereby making identifiers with different cases distinctive. Keywords may be incorporated into identifiers but may not be used alone as identifiers. There are three special identifiers: `main`, `print`, and `play`. They are all function names. `print` and `play` are system-defined functions. `main` is the name of the function that is the entry point for a Tonedef program and needs to be defined in the program.

Examples:

```
good_idt = 1 /* Acceptable identifier */
Good_idt = 2 /* Acceptable identifier, and different than good_idt */

_good_idt = 3 /*  _  is an acceptable starting character for an identifier */
4good_idt = 4 /* Not acceptable identifier because it starts with a number */
```

```
bool = 1 /* Not an acceptable identifier, because boolean is a keyword */
my_bool = 2 /* Acceptable identifier even with keyword incorporated */
```

3.1.3 Operators

Tonedef has a closed set of one and two character operator tokens.

The following are the one-character operators:

+ − / ∗ % = < > : ^

The following are the two character operators:

// ∗∗ ^^ :: @@ >> << == != <= >= || &&

3.1.4 Keywords

There are twenty-one (21) reserved keywords in Tonedef that have semantic meaning in a program and may not be used as identifiers. They are:

Type names:

`int bool string beat pitch note sequence chord phrase rhythm void function`

Control words:

`if else while for foreach in return`

Constant values:

`true false`

3.1.5 Punctuators

Some characters in Tonedef are not operators but have syntactical significance within an expression. None of these characters are allowed in identifiers.

Table 3.1: Punctuators and their Usage in Tonedef

Punctuator	Use
;	Ends a statement
[ ]	Begins and ends a sequence
( )	Expression grouping and function parameter/argument list grouping
,	List operator
{ }	Groups statements into a block
“ ”	Groups characters into a string literal
$	Begins a pitch literal

3.1.6 Literals

In Tonedef, certain sequences of characters are tokenized as literal values of various types.

Integer Literals

An integer literal is any continuous sequence of one or more digits [0-9] that is not part of an identifier.

String Literals

A string literal is comprised of all the characters between quotation marks “. A string literal is opened when a “ character is found and continues until the next ” is met, with the exception that the character sequence \“ has special meaning within the string literal and does not close the string.

Pitch Literals

A pitch literal is a sequence of characters that begins with $ immediately followed by one letter from the set { A, B, C, D, E, F, G}, followed by an optional flat or sharp character from the set { #, b }, followed by a one digit from the set {0 - 9}. There is one special pitch literal that does not meet these rules and is the two-character sequence $_.

3.2 Data Types

Tonedef has 11 data types

```
int bool string beat pitch note sequence chord rhythm phrase void
```

3.2.1 Integers

`int` - Integers are composed of a sequence of one or more digits to represent a whole number. The digits of an Integer may not be separated by Whitespace, and can be negated by placing the unary negation operator “–” before the number. The range of integers is -2^{30} to 2$^{30\text{-}1}$ (or -2^{62} to 2$^{62\text{-}1}$ for 64-bit systems).

3.2.2 Boolean

`bool` - A boolean has two possible values, true or false, which correlate to their respective logical values. Booleans may be cast to an Integer value, with `true` returning a value of 1 and `false` returning a value of 0.

3.2.3 String

`string` - A string is a sequence of 1 or more ASCII characters contained within quotation marks (“...”). Strings may extend across multiple lines of code, but non-explicit Whitespace characters beyond blanks (i.e. tabs and newlines) will not be included in the string. Non-printable characters may be represented in the String by using the following escape sequences.3.2

Table 3.2: String Escape Sequences

Sequence	Description
\“	Double quotation mark
\n	Newline
\t	Tab
\r	Carriage return
\\	Backslash

3.2.4 Beat

`beat` - A beat value represents the duration of a note, where the beat value 1 represents a whole note. More generally, it is a rational number (i.e. can be expressed by `n/d` where `n`, `d` are integers). Beats can be added and subtracted to produce new beat values. There is a special beat-divide operator (//) that produces a beat representing the ratio of the two values of the operands with the left operand being the numerator and the right operand as the denominator.

Examples:

```
beat a = 1; /* a is a whole-note beat */
beat b = 1//4; /* b is a quarter-note beat */
beat c = b + 1//2; /* c is a dotted half-note beat */
beat d = 3; /* d is a 3 tied whole-notes beat */
```

3.2.5 Pitch

`pitch` - A pitch value represents the musical pitch of a note. Pitch values are comprised of exactly one letter from {A, B, C, D, E, F, G}, an optional flat or sharp { #, b } and a one-digit octave number {0 - 9}. Progression of pitches in an octave follow this sequence {C, C#, D, D#, E, F, F#, G, G#, A, A#, B}, where each pitch is a half step above the previous, and this sequence is equal to this sequence of alternate pitch names {B#, Db, D, Eb, Fb, F, Gb, G, Ab, Bb, Cb}. Note that D, F and G do not have an alternate pitch name. This sequence repeats for all the octaves so $B5 is a half step lower than $C6. The octaves are numbered from lowest to highest with C4 being middle C pitch of the traditional music staff. Pitch literals in code must begin with the dollar sign ($) character. Additionally, there is a special null pitch ($_) that does not fall in the ordered progression of pitches and represents no pitch. Pitches form an ordered set can be compared by the comparison operators (==, !=, <, >, <=, >=) with the null pitch being the less than all the lettered pitches.

Examples:

```
pitch pX = $C0; /* pX is the C of octave 0 */
pitch pY = $F#5; /* pY is the F sharp of octave 5 */
pitch pZ = $_; /* pZ is a null pitch */
```

3.2.6 Note

`note` - A note value is an abstraction for musical notes. They are composed of a pitch and a beat. Notes can be constructed from a just a pitch or from a pitch and a beat with the semi-colon operator (:) between them.

Examples:

```
note a = $C0; /* a has a pitch of $C0 and a duration of zero */
note b = $D1 : 1//4; /* b is a quarter-note at pitch $D1 */
note c = pX : bY; /* Notes can be constructed from variables */
```

3.2.7 Sequence

`sequence` - A sequence is a list of integers. They are denoted in code by a comma-separated list of integer values or expressions inside brackets [ ]. Sequences are mainly used to represent changes in pitch through time, or the distance between notes in a chord, where the integer values represent the number of half-step changes in pitch. These distinctions occur when a sequence is used as the right operand of either the (::) or (<<) operators.

Examples:

```
sequence a = [ 0, 2, 4, 5, 7, 9, 11, 12 ];
sequence b = [ 0, 4, 7 ];
chord majorC = ($C4:1) :: b; /* a whole-note C-major chord specifically,
                                          the notes $C4, $E4, $G4 */
phrase majorscaleF = ($F3:1//4) << a; /* an ascending F-major scale*/
```

3.2.8 Chord

`chord` - A chord is a set of zero or more notes that occur simultaneously. A chord can be built empty, built from a single note, or from multiple notes either by adding (+) notes or by the (::) operator.

Examples:

```
chord a; /* empty chord - no notes */
chord b = $G5:1; /* chord containing a single note */
chord c = note_x + note_y + note_z; /* chord containing 3 notes */
chord d = note_x :: [ 0, 7, 12 ]; /* chord containing 3 notes relative
to note_x */
```

3.2.9 Rhythm

`rhythm` - A rhythm represents a series of beats and rests. They are constructed from strings containing only the characters 1, 0, -, “ ” contained within 'single quotes'. The sequence of these characters describes the rhythm where

- 1 signifies the initial playing a note or chord
- - signifies the sustaining of a note or chord
- 0 signifies a rest
- “␣” represents a single “space” and is used to separated the other characters into groups

Each group represents a total duration of a whole note so the size of the groups determine the time of each character in that group. The group sizes should be 1, 2, 4, 8, 16, which correspond to each character having length of a whole-note, a half-note, a quarter-note, an eighth-note and a sixteenth- note, respectively. If a group of characters is not equal to any of these values, the first n characters that form a complete group are used and the rest ignored.

Examples:

```
rhythm a = '1 11 1111'; /* one whole-note, followed by 2 half-notes,
                                              followed by 4 quarter notes */
rhythm b = '1--- 1-1- 1111'; /* b is equal to a but written      differently
                                                     as a string */
rhythm c = '10101010 1111000011110000'; /* c is (an eighth note then eighth
                              rest) x 4, then (4 sixteenth notes,
                              4 sixteenth rests) x 2 */
```

3.2.10 Phrase

`phrase` - A phrase is an ordered collection of chords, where the ordering represents time. A phrase can be built from a single chord, from other phrases using the append phrases (@@) or combine phrases (**) operators, or using the apply operator (<<). The musical sequence represented by a phrase value can be played by sending the phrase to the `play` function.

Examples:

```
phrase a; /* empty phrase */
chord x = ($D3:1) :: [ 0, 4, 7 ];
phrase b = x; /* phrase that contains just chord x */
phrase c = b @@ b; /* c is chord x played twice */
phrase d = $E5 << [ 0, 2, 2, 4 ] << '11 0 11'; /* phrase built using the
```

3.2.11 Void

`void` - Void is a special type for functions that return no value. Non-function identifiers cannot have type void.

3.2.12 Type Conversion

Tonedef will perform certain conversions of types when values do not exactly meet the expected type of a function argument or operand. The following table shows the allowed conversions:

Table 3.3: Type Conversions

Type Conversion	Description
bool → int	true → 1 and false → 0
int → bool	0 → false and everything else becomes a true value
beat → int	the value of beat is rounded down to an integer
int → beat	the beat is a rational number with the int as the numerator and 1 as the denominator
pitch → note	pitch is promoted to a note with duration = 0
beat → note	beat is promoted to a note with that duration and pitch = $_
note → pitch	note is demoted to only its pitch value
note → beat	note is demoted to only its beat value
note → chord	note is promoted to a chord that contains just that one note
chord → phrase	chord is promoted to phrase that contains just that chord

3.3 Declarations

Declarations are used to create new variables within a block of code of specified type and optionally initialize their values.

3.3.1 Declaration Syntax

A declaration is an expression in one of the two following forms:

```
        type identifier
        type identifier = initialization-expression
```

Where

- `type` is one of the type keywords: `int`, `bool`, `string`, `beat`, `pitch`, `note`, `chord`, `sequence`, `phrase` or `rhythm`.
- `identifier` is a non-reserved alpha-numeric sequence as described in Section 3.1.2
- `initialization-expression` is any legal Tonedef code that returns a value of agreeable type with the declaration.

3.3.2 Blocks

A block is a section of code enclosed by braces { }. Blocks can be nested within other blocks. Identifiers visible in an outer block are visible in the inner block, but identifiers declared in the inner block will not be visible in the outer block when the inner block ends.

Examples:

```
void function f ()
{ /* start of a block for this function */
        int x = 0;
        while ( x < 10 )
        { /* start of a sub-block */

        } /* end of the sub-block */
} /* end of the block for function f */
```

3.3.3 Scope

The scope of an identifier is the subsequent statements within the block of code where it is declared including sub-blocks of that block. Declarations can appear after certain keywords that open a block of code. These keywords are `function`, `for`, and `foreach`. When identifiers are declared in these expressions, the scope of the identifiers is the block opened by the keyword. Scope does not extend to the execution of function calls. At the beginning of a functions execution, its parameters will be the only identifiers in scope.

Examples:

```
void function f ( phrase p ) {

        /* p is in scope */
        foreach ( chord c in x) {
                /* c and p are in scope */
                note n = some_other_function();
                /* c, p, and n are not in scope while
                        some_other_function executes */
        }

        /* c and n are no longer in scope, but p still is */
        for ( int i = 0; i < 0; i = i + 1){
        /* p and i are in scope */
        }

        /* p is only identifier in scope */
}
```

3.3.4 Identifier Naming

All identifiers within a block of code must be unique and a sub-blocks identifiers must not conflict with the identifier names in its parent block. This means that an identifier is visible over its entire scope and cannot be hidden by a subsequent re-declaration of the identifier.

Examples:

```
void function f ( note x ) {

        chord x; /* this is NOT legal because x is already an identifier
                                in this block */
        for ( int i = 0; i < 5 ; i = i + 1 ){ }
        for ( int i = 5; i > 0 ; i = i -1 ){ }

        /* this re-use of identifier i is legal because the first i is no
        longer in scope when the second is declared */
}
```

3.4 Functions

A Tonedef program consists of a collection of functions. Functions are re-usable segments of code that can be invoked from within the code of other functions. This provides a way to break up a program into smaller tasks. Each function consists of a name, a list of parameters, a return type and a block of execution code.

3.4.1 Function Declaration/Definition

A Tonedef function is declared like this:

return_type `function` function_name (parameter_list) { code }

Where

- `return_type` is Tonedef data type that will returned from the function.
- `function` is a keyword in Tonedef and must be included in every function declaration.
- `function_name` is a name for the function that will be used to reference it throughout the program.
- `parameter_list` is a list of zero or more comma-separated parameter declarations where each declaration is of the form `type identifier`.
  - No two parameter names are the same within one function, and no parameter name is the same as its function's name
- `code` is a possible empty sequence of Tonedef code to be executed when the function is called.

Functions cannot be declared within other functions. They must all be declared at the program level. Function names are visible throughout the program where they are declared, meaining that a function can be called from any other functions execution code and that the ordering of the function declarations within a program is inconsequential.

3.4.2 Function Calling

A function call is an expression of the following form:

function-identifier (arguments-list)

Where

- `function-identifier` is a name corresponding to some user-defined or system-defined function.
- `arguments-list` is a possibly empty and comma-separated list of arguments to pass that function.
- Each argument is an expression that resolves to a value (i.e. not void).
- The number, order, and types of the argument values match the parameter types declared in the function declaration.

All argument expressions are resolved before the function call is executed. An argument expression can itself be a function call so function calls within the same statement will be executed inside out.

Examples:

```
fn_a(fn_b(5)); /* fn_b (5) is executed and its return value is passed as the
                                      argument to the execution of fn_a */
```

3.4.3 Main

Each Tonedef program must include a definition of a function named `main` with type `int` or `void` and no parameters. This function is the entry point for execution of the program.

3.4.4 Play

The function `play` is a system-defined function of type void that takes one parameter of type `phrase`. When called, this function produces audio output from the musical expression represented by the phrase argument passed to it.

3.4.5 Print

The function `print` is a system-defined function of type void that takes one parameter of type `string`. When called, this function writes its argument string to the output console.

3.5 Operators and Expressions

Tonedef operators are any of a closed set of one and two character sequence described in Section 3.1.3 relating to Operators. Some operators are unary and take an operand on the right side. Others are binary and take an operand on both the left and right side. The following sections specify the types of operands the various operators take and the types they return.

Tonedef expressions are sequences of literals, identifiers, punctuators and operators, which evaluate to a value of a Tonedef type. The allowed sequencing of these elements is explained in the following section.

3.5.1 Expressions

Identifiers and Literals

An expression can be any literal or non-function identifier. These expressions evaluate to the value of the literal or the value bound to the identifier. This defines the following syntax rule:

```
expression :=
        literal
        identifier
```

Operators and Function Calls

Any operator or function identifier along with the appropriate operand or argument expressions combines to be an expression. These expressions evaluate to the result of the operation/function. This adds another syntax rule:

```
expression :=
        <unop> expression
        expression <binop> expression
        function_identifer ( expression list )
```

Parentheses

An expression within parenthesis evaluates to the same value and type as the expression without parentheses. Parentheses can be used to change the precedence of operators within an expression.

```
expression :=
        (expression)
```

3.5.2 Arithmetic and Boolean Operators

Comparison Operators (==, !=, < <=, >, >=)

The types of the two operands for comparison operators must be the same. Only operands of type `int`, `beat`, `pitch` and `note` are allowed.

expr1 == expr2
Returns the Boolean value `true` if expr1 has the same value as expr2, and `false` if the values are different.

expr1 != expr2
Returns the Boolean value `false` if expr1 has the same value as expr2, and `true` if the values are different.

expr1 < expr2
Returns the Boolean value `true` if the value of expr1 is less than the value of expr2, and `false` if otherwise.

expr1 <= expr2
Returns the Boolean value `true` if the value of expr1 is less than or equal to the value of expr2, and `false` if otherwise.

expr1 > expr2
Returns the Boolean value `true` if the value of expr1 is greater than the value of expr2, and `false` if otherwise.

expr1 >= expr2
Returns the Boolean value `true` if the value of expr1 is greater than or equal to the value of expr2, and `false` if otherwise.

Multiplicative Operators (*, /, %, //)

These operators take operands of type `int` or `beat`.

expr1 * expr2
Multiplies expr1 with expr2 and returns the result.

expr1 / expr2
Divides expr1 by expr2 and returns the integer result (rounding towards zero). Will return an error if expr2 is a null or zero value.

expr1 % expr2
Yields the remainder of expr1 divided by expr2. Will return an error if expr2 is a null or zero value.

expr1 // expr2
Divides expr1 by expr2 and returns the `beat` result. Will return an error if expr2 is zero valued. Both expressions must be of type `int` or `beat`, noting that beats will first be rounded to ints before the new beat is created.

Additive Operators (+, -)

These operators can take operands of type `int` or `beat` as described here. The operators are overloaded for other types of operands, but the descriptions of those are in Section 3.5.3 on Musical operators.

expr1 + expr2
Add expr1 to expr2 and return the result. Both expressions must be of the same type or can be cast to equivalency.

expr1 - expr2
Subtract expr2 from expr1 and return the result. Both expressions must be of the same type or can be cast to equivalence.

-expr
The result is the negative of expr, and has the same type.

Boolean Logic Operators (&&, ||, !)

These operators take operands of type `bool`.

expr1 && expr2
Logical AND on two boolean expressions. Returns `true` only if both expressions are true.

expr1 || expr2
Logical OR on two boolean expressions. Returns `true` if at least one of the expressions is true, and `false` only if they are both false.

!expr
Logical NEGATION. Returns `true` if the expression is false, and `false` if the expression is true.

3.5.3 Musical Operators

Raise Note/Pitch by Steps (ˆ)

```
np ^ x
```

Returns a new note or pitch that is a copy of the left operand (`np`) that is raised (or lowered for negative `int` values) by `x` half-steps. The left operand can be either of type `note` or `pitch`, and the right operand is of type `int`.

Raise Note/Pitch by Octaves (ˆˆ)

```
np ^^ x
```

Returns a new note or pitch that is a copy of the left operand (`np`) that is raised (or lowered for negative `int` values) by `x` octaves. The left operand can be either of type `note` or `pitch`, and the right operand is of type `int`.

Note Creation from Pitch and Beat (:)

```
pitch : beat
```

Returns a new note that has a pitch equal to the left operand(`pitch`) and beat equal to the right operand (`beat`).

Chord Creation from Note and Sequence (::)

Returns a new chord that is built relative to the left operand (`note`) according to the integers in the right operand (`sequence`). Each integer in the sequence is a number of steps to raise/lower the pitch of note before adding to the resultant chord. The note itself is only added to the chord if 0 is one of the integers of in the sequence. The order of integers in the sequence does not matter for this operator because all integers are interpreted relative to the pitch of the note.

Examples:

```
chord a = ($C4:1) :: [ 0, 4, 7 ] ;  /* a has the notes $C4, $E4, and $G4 with
                                          whole-note duration */
chord b = ($E5:1) :: [ 1, -2, 7 ] ; /* b has notes $F5, $D5, $B5 */
```

Addition of Notes/Chords (+)

```
nc1 + nc2
```

Returns a new chord that contains the notes of the two operands (`nc1`, `nc2`) combined. Each operand can be of type `note` or `chord`.

Sequence Application (<<)

```
note << sequence
```

Returns a phrase that is built by applying the right operand (sequence) as a series of pitch increases/decreases relative to the pitch of the left operand (note). To include the note in the phrase, the sequence should contain a 0. The phrase will have n notes where n is the number of integers in sequence, and each note in the phrase will have the same duration as the beat of the left operand.

Examples:

```
phrase a = ($C4:1//4) << [ 0 , 4, 7, 12 ]; /* a has the notes $C4, $E4, $G4,
                                                $C5 played as quarter-notes in succession */
```

Rhythm Application (<<)

```
phrase << rhythm
```

Returns a new phrase that has the same chords and order of chords as the left operand (`phrase`) but with modified beats and locations in time according to the right operand (`rhythm`). The ones in the string representation of rhythms are the locations in time that the non-empty chords of phrase are put at in the new phrase. If the phrase has more chords than the number of ones in rhythm, then those extra chords are omitted from the resultant phrase. If the phrase has fewer chords than the number of ones in rhythm, then those extra time locations are filled with rests.

Examples:

```
phrase a = ($C4:1//4) << [ 0 , 4, 7, 12 ]; /*same as above example */
phrase b = a << '1 11 1'; /* the four notes of a are in b, but with new
                                                    durations - whole, half, half, whole */
phrase c = ($D3:1) << [0,1,2,3,4,5,6,7,8,9,10,11,12] << '1110111011101110 1';
                /* c is a chromatic scale starting at $D3 played in 16th notes with 16th r
                every third note, and ending on a whole note at $D4 */
```

Shift Phrase Right (>>)

```
phrase >> beat
```

Returns a new phrase that is equal to the left operand (`phrase`) with x beats of rest added to the front, where x is the right operand (`beat`).

Examples:

```
phrase d = c >> 1//2 ; /* d is the chromatic scale from the above example
                        with a half note of rest at the begging */
```

Combine Phrases (**)

```
phrase1 ** phrase2
```

Returns a new phrase that is the two operands (`phrase1`, `phrase2`) merged together. The start of each phrase is aligned and chords of the resultant phrase are created from the chords of `phrase1` and `phrase2` at each moment in time in the phrases. This operator is commutative so the order of the 2 operands does not change the result. If one operand phrase is longer than the other is, then the shorter one is interpreted to have beats of rest its notes during this merging process. The resultant phrase has total duration equal to the maximum of the durations of the two operands.

Combine Phrases (@@)

```
phrase1 @@ phrase2
```

Returns a new phrase that is the left operand (`phrase1`) followed by the right operand (`phrase2`). There is no overlap of notes from each operand. This operator is not commutative because the order of the operands determines the ordering of their notes in the resultant phrase. This operator is left associative. The resultant phrase has total duration equal to the sum of the durations of the two operands.

3.5.4 Assignment

```
identifier = expression
```

Tonedef has one assignment operator (=), which takes an `identifier` as the left operand and an `expression` as the right operand. This operator evaluates the expression to a value and binds that value to the name identifier. The type of the value of expression must match the type of identifier or be a type that can be converted to the type of identifier. This operator also returns the value of expression and is right associative. Assignment is done by value so two identifiers can be equal to the same value, but not bound together such that re-assigning one changes the other.

Examples:

```
pitch a = $C4;   /* a is $C4 */
pitch b = a;     /* b is $C4 as well */
a = b ^ 4 ;              /* a is now $E4, and b remains $C4 */
pitch c = b = a;  /* a, b, c are all now $E4 */
```

3.5.5 Operator Precedence

The following table shows the order of precedence of Tonedef operators along with the associativity (order of evaluation) of each level. The top of the table is the highest precedence.

Table 3.4: Precedence and Associativity of Operators

Operators	Associativity
-(unary) !	right to left
* / % //	left to right
^ ^^	left to right
:	left to right
:: << >>	left to right
**	left to right
@@	left to right
+ - (binary)	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
=	right to left
,	left to right

3.6 Statements

This section describes the different types of Tonedef statements. Every function definition is a sequence of statements. Statements are executed in the order they appear in a function body.

3.6.1 Compound Statements

Statements are grouped together into a block using braces { }. This allows a sequence of statements to be treated as a single statement. All previously visible identifiers remain visible within the block (see Section 3.3.2 - Blocks). Such a grouping creates a new level of scope where new variable declarations inside the block are only in scope within that block (see Section 3.3.3 - Scope). Compound statements are useful following any of the control structures listed in this section.

3.6.2 Expression Statements

Any valid expression can be used as a statement by following the expression with a semicolon(;).

```
expression;
```

3.6.3 If Statement

The if statement is a control structure for conditional execution of code. An if statement has the following syntax:

```
if ( expression ) statement1 else statement2
if ( expression ) statement1
```

When the `expression` is true (i.e. evaluates to a boolean value of `true` or non-zero `int` value), `statement1` is executed; otherwise, `statement2` is executed. The `else statement2` portion of the if-statement is optional, in which case nothing is executed if the `expression` is false.

3.6.4 While Statement

The while statement is a control structure for looping execution of code as long as a control expression is true. A while statement has the following syntax:

```
while ( expression ) statement
```

The `expression` is evaluated before the potential execution of the `statement`. If the `expression` is true, the `statement` is executed, then this two-step process repeats. If the `expression` is false, the `statement` is not executed and the while loop terminates. The `statement` should perform some computation that eventually causes the `expression` to be false and terminate the loop. Otherwise, the loop will infinitely repeat.

Example:

```
note x = $C4 ;
while ( x < $C5 ) {
        x = x^1;
}
```

In this example, `x` is a note and is raised one-step. Since $C5 is a higher pitch than $C4, the loop eventually terminates. If the loop body were `x = x^-1`, then x would be decreasing in pitch and the loop would not terminate.

3.6.5 For Statement

The for statement is a control structure for iterative execution of code. A for statement has the following syntax:

```
for ( expression1 ; expression2 ; expression3 )  statement
```

The `expression1` is executed as a statement once at the beginning of execution of the `for` statement. Then, `expression2` is evaluated. If it is true, statement is executed, then `expression3` is executed as a `statement`, then this process repeats. If expression2 is false, the loop terminates.

A for statement is equivalent to the following while statement:

```
expression1;
while (expression2) {
        statement
        expression3;
}
```

Like `while` statements, a `for` statement needs a control expression (`expression2`) to eventually be false to terminate. This is normally done in `expression3`, but can be done in `statement` as well.

```
beat b = 0;
for ( int i = 1 ; i < 10 ; i = i + 1) {
        b = b + i // 4 ;
}
```

In this example, the body of the loop does not modify `i`, but `expression3` does, so `expression2` is eventually false. Specifically, this body is executed 9 times and `b` equals the length of 45 quarter-notes.

3.6.6 Foreach Statement

The foreach statement is a control structure for iterating through the elements of one of Tonedefs ordered data types (sequence, phrase). A foreach statement has the following syntax:

```
        foreach ( type identifier in expression ) statement
```

Where

- `type` can be only `chord`, `note`, or `int`
- when `type` is `chord`, `expression` must evaluate to a phrase value
- when `type` is `note`, `expression` must evaluate to a chord value
- when `type` is `int`, `expression` must evaluate to a sequence value

The foreach statement begins with evaluating expression once to a phrase or sequence value. Each iteration of the loop begins with `identifier` becoming the next element in this value. Then, the statement is executed.

```
/* phrase p defined previously */
phrase z;
foreach ( chord c in p ) {
        z = c @@ z;
}
```

In this example, the `foreach` loop executes once for each chord in the phrase and builds a new phrase `z` that is the reverse of `p`.

3.6.7 Return Statement

The return statement terminates execution of a function and has the following syntax:

```
return expression;
return;
```

The `expression` is evaluated to a value. This value needs to have the same type as the return type of the function (or be a type that can be promoted/demoted to the return type). At this point, the execution of the function ends and this value becomes the return value of the function call. Program execution continues from the location of the function call.

A function may contain multiple return statements within its body and/or sub-blocks in its body. However, a function (except for void typed functions) must contain at least one return statement at the outer most block of its body. A function of type void should use the expression-less return statement syntax, or use no return statements, in which case it will return when it reaches the end of its body.

```
pitch function f ( sequence s , pitch p) {
        foreach ( int i in s ) {
                if ( i == 0 ) { return p ; }
                else { p = p ^ i ; }
        }
        return p;
}
```

In this example, there is a `return` statement in a conditional statement in a looping statement. This allows the function to terminate in the middle of the loop; however, the second `return` statement is needed because there is no guarantee that the if condition is ever true or even that the sequence has any elements to iterate through.

Chapter 4

Project Plan

4.1 Planning, Specification, Development and Testing Process

After forming a team and each team member proposing different topics, the team easily made a unanimous decision to work on a programming language to simplify music creation and definition. Due to the fact that the concept of the language was proposed by Curtis Henkel and drew from his experience in music, the team declared him the captain. We then worked on defining desired features and functionality in the programming language we would like to have, followed by drawing out the basic types, operators, keywords and language constructs we would like to have. This enabled us to propose a well though out language and Language Reference Manual.

For development, the team utilized a fast iterative development cycle where small incremental changes were made, tested and then checked in. Team members would complete tasks as they were required, allowing for the most important parts of the project be completed earliest and everyone to contribute to its completion. While certain members specialized in certain areas of the implementation, this approach ensured everyone was involved and understood how everything worked.

For testing, the team utilized a simple testing suite written in Python. For each piece of completed functionality, the team would write a unit test that would compare the actual output against the expected output for a given input. As the project progressed, all tests in the test suite would be run to ensure that no functionality was broken with the latest changes.

4.2 Programming Style Guide

Our team adopted a custom programming style guide adapted from the following resources:

- Caml programming guidelines (`http://caml.inria.fr/resources/doc/guides/guidelines.en.html`)
- CS20a OCaml Style Guide (`http://www.cs.caltech.edu/~cs20/a/style.html`)

When not explicitly defined, our programming style defaults to the Caml programming guidelines. Below are the guidelines utilized in our project:

- Be simple and readable
- General
  - A space should always follow a delimiter symbol, and spaces should surround operator symbols.
  - A tuple is parenthesized and the commas therein (delimiters) are each followed by a space: (1, 2)

- Indent long character strings with the convention in force at that line plus an indication of string continuation at the end of each line (a \ character at the end of the line that omits white spaces on the beginning of next line)
- Use parenthesis to simplify expressions for readability

- Indentation
  - The indentation between successive lines is 2 spaces
  - Tab stops shall not be used within the program
  - Individual lines of code should not be longer than 80 characters
  - The expression following a definition introduced by let is indented to the same level as the keyword let, and the keyword in which introduces it is written at the end of the line
  - All the pattern-matching clauses are introduced by a vertical bar, including the first one
  - All pattern matching clauses are indented as normal to the start of the construct
  - In if ... then ... else ..., if the conditions and expressions do not fit, branch into multiple lines and indent as normal
- Comments
  - Comment complex functions and pieces of code
  - Do not comment unless necessary
  - Avoid comments in the bodies of functions, unless functions are complex and require commenting midway
  - Multiline comments must be proceeded with a $*$ after the first line and aligned with the $*$ of the first line
- Naming
  - Use simple case for variables
  - Separate words in names by underscores
- Whitespace
  - No trailing whitespace at the end of lines
  - Empty lines should not contain whitespace

4.3 Project Timeline

The proposed project timeline is presented in Table 4.1.

Table 4.1: Project Timeline

Date	Description of Event
9/13/2010	Team formulated
9/13/2010	Subversion repository created
9/20/2010	Project concept finalized
10/17/2010	Scanner completed
10/17/2010	AST completed
10/21/2010	Java JFugue samples completed
11/02/2010	Parser completed
12/04/2010	External Java JFugue program completed
12/10/2010	Bytecode definition completed
12/14/2010	Compiler completed
12/14/2010	Tonedef phrase to JFugue music string conversion completed
12/18/2010	Bytecode interpreter completed
12/19/2010	Sample programs completed
12/23/2010	Consistent code styling completed

4.4 Roles and Responsibilities

All team members generally took part in the development of all the project components. However certain members took leads or a more involved part in certain components, as defined by the following breakdown:

- Curtis Henkel (Team Leader)
  - Parser
  - Semantic analysis in compiler
  - Bytecode interpreter
- Matthew Duane
  - Abstract syntax tree
  - Translation, control flow and operations in compiler
  - Unit tests
- Kevin Ramkishun
  - Creating the testing suite and tests
  - Build environment and repository administration
  - Bytecode design, implementation of binary operators in compiler
- Chatura Atapattu
  - Researching JFugue, its use and writing JFugue programs and samples
  - Converting Tonedef phrases to JFugue music strings
  - Report and code styling

4.5 Software Development Environment

The Tonedef language was implemented completely in OCaml, utilizing Java and JFugue (a Java API for Music Programming) to play phrases created in Tonedef. The Java SDK utilized was Oracle's Java 6 Update 21 JDK. The JFugue (`http://www.jfugue.org/`) API utilized was version 4.1.0 Beta. For source control, the team utilized Subversion, which was hosted by Assembla (`http://www.assembla.com/`). As OCaml is most native to a Unix environment, the team utilized Make for automating the build process. A simple test suite was also written in Python 2.6.6.

Each team member utilized various development environments and tools during the course of the project:

- Curtis Henkel
  - Mac OS X
  - XCode
  - Terminal
- Matthew Duane
  - Windows and Ubuntu (VirtualBox)
  - Notepad++ and Geany
  - Cygwin and Terminal
- Kevin Ramkishun
  - Mac OS X
  - Emacs (with OCaml mode from Jane Street Capital)
  - Terminal
- Chatura Atapattu
  - Windows and Ubuntu (VirtualBox)
  - Ubuntu (natively)
  - JCreator, Eclipse, & GEdit
  - Terminal

Documentation was written collaboratively using Google Docs. This report was written with the LaTeX typesetting system, collaboratively, using the subversion repository mentioned above.

4.6 Project Log

The project log is included in Appendix C.

Chapter 5

Architectural Design

The Tonedef compiler can be best described from the block diagram in Figure 5.1. In this diagram, it is visible that the Tonedef source code first goes through the scanner, which performs lexical analysis and gives tokens. These tokens then enter the parser which performs syntactic analysis and, with the help of our abstract syntax tree interface, produces an abstract syntax tree. The compiler then performs semantic analysis and produces intermediate representation, which is then handed off to the bytecode executer. The bytecode executer maintains a data stack and executes all of the bytecode, printing messages as necessary and utilizing JFugue to play phrases (after converting them to JFugue-style music strings).

5.1 Component Breakdown

This section details the functionality of each source code file that comprises the Tonedef compiler.

tonedef.ml

Coordinates all of the sub-components, feeding the output of one into next

Scanner.token:	char lexbuf $\rightarrow$ Parser.token lexbuf
Parser.program:	Parser.token lexbuf $\rightarrow$ Ast.program
Compiler.translate:	Ast.program $\rightarrow$ Bytecode.bprogram
Execute.run:	Bytecode.bprogram $\rightarrow$ ()

scanner.mll

Defines regular expressions to recognize each token in the language. Converts the source code of a Tonedef program from a character stream to a token stream.

ast.mli

Defines types for the Abstract Syntax Tree representation of a Tonedef program.
Types: `uop, bop, literal, td_type, typed_id, expr, stmt, func, program`

parser.mly

Defines the syntax rules for the Tonedef language. This is the context-free grammar for the language. Converts the token stream into an Abstract Syntax Tree

bytecode.mli

Defines the bytecodes for the bytecode representation of a Tonedef program. Defines types for representing a stack of memory to hold program state during execution.
Types: `bytecode, bprogram, mnote, mchord, mphrase, msequence, memory, stack`

compiler.mli

Performs semantic analysis on a Tonedef program – checks that the naming, scoping rules are not violated, checks that expression resolves to types that are valid for each assignment, function argument, and operator. Then translates the AST representation of a Tonedef program into a bytecode array.

execute.ml

Maintains a stack representing the state of the Tonedef program, a stack pointer, a frame pointer, and a program counter. Processes individual bytecode operations from the output of the compiler and updates the program state accordingly. Performs system calls to `PlayJFugue` in order to produce audio when specified by the execution of the bytecode. Also has functions for translating a Tonedef phrase into a JFugue string (so that `PlayJFugue` can play it).

printer.ml

Converts various data types to strings, and prints the abstract syntax tree of a Tonedef program.

helper.ml

Implementation of functions used by other components in the system.

is_promotable_to	Checks if one type can be converted to another
get_signature_map	Foreach function in a tonedef program, builds a list of tonedef types representing the signature of a function [return type; arg1_type, arg2_type ...] and returns a `StringMap` mapping function names to their signature lists
binop_check_operand_types	Checks that the types of the operands on a binary operator are allowed in the Tonedef language
checkfunction	Checks that identifier names are declared and used properly according to the scope rules within a function declaration and returns a list of local identifiers used in the function
string_of_bc	Converts a bytecode into a string to be printed
int_of_pitchstring	Converts a pitch string into an integer representation (for comparing, adding, etc).

PlayJFugue.java

Takes a string as an argument and then uses the JFugue Java library (www.JFugue.org) to play audio represented by that string. This component is invoked during execution of a program.

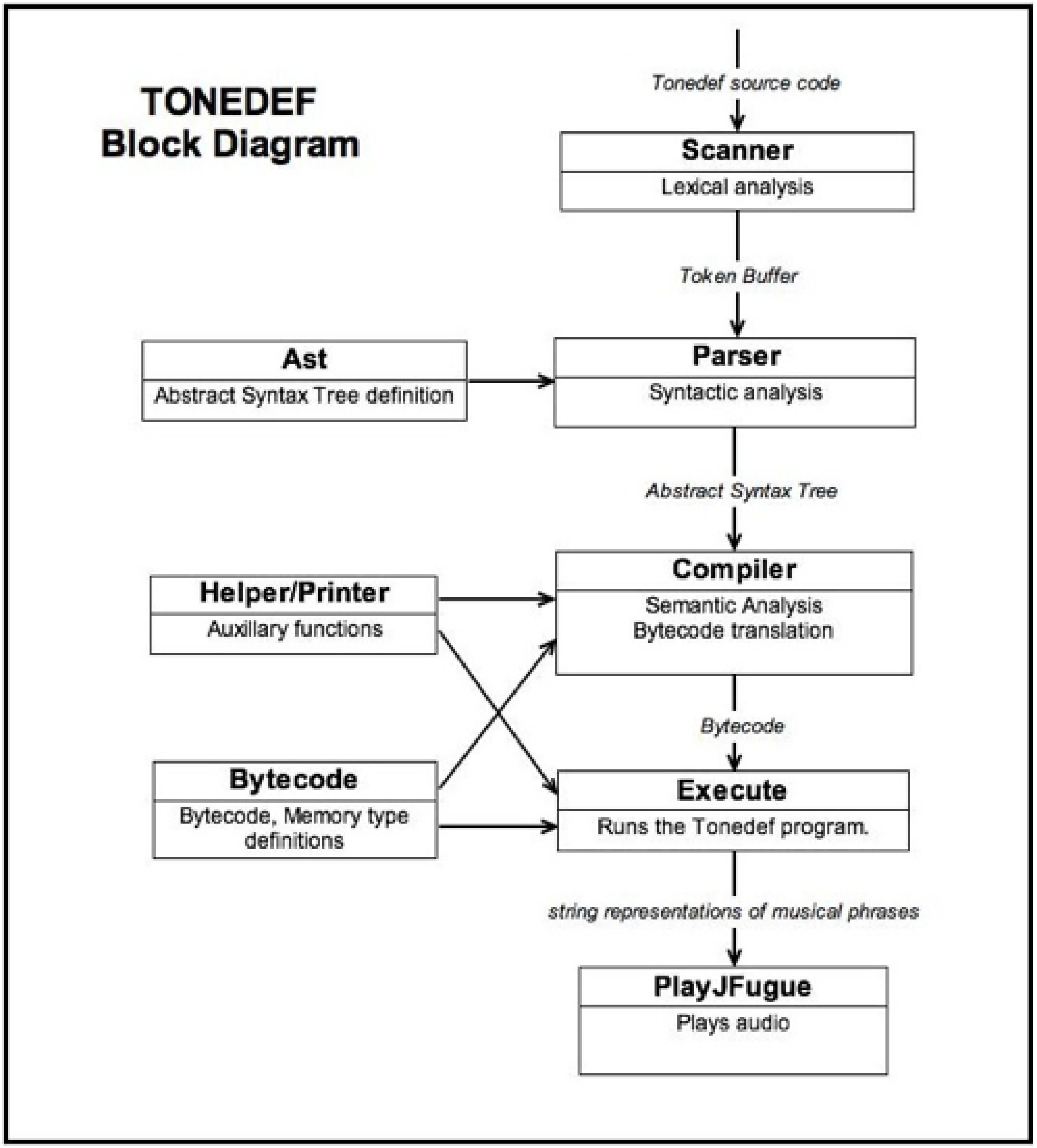

Figure 5.1: The Tonedef Compiler block diagram

5.2 Component Implementation

Although many of the responsibilities overlapped when producing source code and documentation, Table 5.1 attempts to best categorize the role that each team member played in the developing of Tonedef.

Table 5.1: Component Implementation

Component(s)	**Implementer(s)**
scanner.mll	Kevin Ramkishun
ast.mli/parser.mly	Curtis Henkel
printer.ml	Matthew Duane
compiler.ml/helper.ml	Matt Duane - Translation from AST to bytecode Curtis Henkel - type/scope/name checking
bytecode.mli/execute.ml	All members - each implemented different execution cases
PlayJFugue	Chatura Atapattu
tonedef.ml	Kevin Ramkishun & Matt Duane
Tests, Makefile	All Members

Chapter 6

Test Plan

One of the organizational goals that was given high priority early on in the language implementation process was to have a very solid testing framework in place from the beginning. Learning from past examples and the experiences of our mentors, we understood that a good testing framework can guarantee that new features and commits will not interfere with existing functionality. Professor Alfred Aho, for example, stated that one of the things that he wished he did differently when implementing *Awk* was having a regression test program in place earlier.

It was found that the most helpful tool to have to test our language was programs that were representative of the types of applications that would be written in our language. Good sample programs aided us throughout the entire development & testing cycle.

Very early in the development process we relied on manual testing to get a bare-bones compiler in place. After the Scanner and Parser had been built, an automated regression test suite was put into place and relied on heavily.

By the end of the development cycles, when all of our tests were being passed, we wrote several "demo" programs as representative samples of what our language could accomplish. These source programs are located in Appendix B, and were used as final tests. If our sample programs (which implemented rather complicated algorithsm, like quicksort) could compile and run correctly, then our target has been met.

6.1 Early Testing of Parser and Scanner

During the initial phases of implementation (specifically the implementation of the Scanner and the Parser) testing was done using a "pretty-printer." Our initial test suite comprised of a few realistic programs that covered the entire grammar of our language. These programs were written from scratch for the sole purpose of testing the Scanner and Parser.

6.1.1 Testing the Scanner

At the heart of Lexical Analysis test was a "pretty-printer" which took a stream of tokens as an input and output the code that likely produced that string of tokens. Although this test could not preserve whitespace[1], we were able to compare the output of this test with the input of the test by simply ignoring whitespace.

To test the scanner, we fed our programs to the scanner for tokenization and then piped the tokens into the pretty-printer. We then visually compared the output of the pretty-printer to the original source code and

[1]Whitespace is lost during the tokenization process, because it is used to separate tokens

noted any discrepancies. The most common failure case for this test was having a token be lost. This is likely due to the simple nature of the Scanner.

Since our test programs for the scanner covered the entire grammar of the language, all issues with the scanner could be solved with this simple test.

In the later stages of development of the language, the Scanner would be inherently tested with every regression test. This is because the scanner must be functionally correct for any other part of the compiler also be correct.

6.1.2 Testing the Parser

The parser was tested using the same method as the scanner. This time, however, grammatically incorrect statements in our language would be flagged as a parse error and would not make it through the compiler. If a program was a member of our grammar, then it would make it through both the scanner and the parser and would be output using the pretty-printer.

The goal with testing the Parser was to come up with as many patterns as possible. In addition to testing good patterns, we came up with just as many failing patterns, to make sure that our context-free grammar that was at the heart of the parser was correctly designed.

In the later stages of development of the language, the Parser would be inherently tested with every regression test. This is because the parser must form a correct abstract syntax tree for any other part of the compiler (ex. creating intermediate representation) to work properly.

6.2 Regression Test Suites

After the Scanner and Parser were completed and the grammar of our language was finalized, work began on the actual compiler (which generated intermediate representation in the form of Tonedef bytecode) and the executer (which maintained a stack and interpreted the intermediate representation). This more mature stage of development called for a more mature method of validating all code that was committed to the project repository.

By analyzing the feature set of Tonedef, we decided on the Test Suite structure shown in Table 6.1. This structure corresponds to the directory structure of the tests in our code repository.

Table 6.1: The Structure of the Tonedef Test Suite

Name	Purpose
Top Level	General Purpose Tests of Grammar
Suite_Arithmetic	Tests for arithmetic on integers, beats, pitches
Suite_Logic	Tests for logical operators, conditionals, booleans
Suite_Ops	Tests for binary operators (ex. creating notes, pitches, chords)
Suite_Strings	Tests on strings and string operators, printing

A plan was put into place that any committed feature would be accompanied with at least one test in the test suite. A test consists of a Tonedef source file that uses the feature and produces some sort of output (using the print statement), and a "Gold" file that contained the expected output. Each individual test should only test one specific feature, so that if it fails, it is obvious what functionality has failed. If a test contained more than one feature and failed, more work would have to be done to see which feature in that test was the culprit.

At the completion of Tonedef development, the test suite contained 35 individual tests.

6.3 Automated Testing

Accompanying our test suite was a method to automatically perform all tests at compile time. This was implemented using a script written in the Python programming language. This language was chosen because we were familiar with it and it is included on all good, modern Operating Systems. The script was named `test.py` and was put in the root directory of our repository (along with the Makefile).

The script recurses through the test directory, looking for files with the `.tdf` extension. It then finds the `.gold` file with the same name as the `.tdf` file. The script executes the Tonedef executable, piping the `.tdf` file as an input, and reading the output. It then compares the output to the `.gold` file.

For each `.tdf` file found, the script will issue either a

- Warning, if the `.gold` file could not be found for the `.tdf`
- Fail, if the `.gold` file does not match the output of Tonedef on the `.tdf`
- Pass, if the `.gold` file matches the output of Tonedef on the the `.tdf`
- Error, if Tonedef produces an error while running the `.tdf`

If an error occurs, the script prints the error. In addition, the script provides the `-v` switch for verbose mode, which will print out tonedef's output and the .gold file for visual comparison. The script enumerated the number of errors, warnings, passes, and failures and reported them at the end.

The `.gold` files were created manually; the test cases were kept so simple that the output of them should be obvious.

`test.py` was eventually incorporated into our Makefile to be run every time the project was compiled. No new code that failed our tests was committed to the repository without first getting responses from all group members.

The full Python script is presented in the appendix Section A.1 and its output at the time of writing is presented in Figure 6.1.

The inclusion of our Test Suite was one of the most important decisions that we made during the development of Tonedef. It allowed us to immediately catch errors that broke existing code on numerous occasions, and gave us peace of mind when committing changes to the code. With a code base as large as Tonedef, this type of testing framework is absolutely necessary.

Automated testing made this an extremely pleasant experience because after we had written the test script, there was no additional work to be done. Writing test cases took very little time, but the benefit that they gave us was tremendous. We attribute much of our successful implementation of Tonedef to the test suite.

```
--------------------------
Automatic Regression Tests
--------------------------
PASS: tests/hello_world.tdf
PASS: tests/simple_var.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_01.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_02.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_03.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_beat.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_mod.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_sub.tdf
PASS: tests/Suite_arithmetic/arithmetic_test_unop_01.tdf
PASS: tests/Suite_logic/comparison_ops_types.tdf
PASS: tests/Suite_logic/comparison_test_01.tdf
PASS: tests/Suite_logic/logic_test_01.tdf
PASS: tests/Suite_logic/suite_logic_test_01.tdf
PASS: tests/Suite_logic/suite_logic_test_02.tdf
PASS: tests/Suite_logic/suite_logic_test_andor_01.tdf
PASS: tests/Suite_logic/suite_logic_test_for01.tdf
PASS: tests/Suite_loops/loops_test_01.tdf
PASS: tests/Suite_ops/append_phrases_test_01.tdf
PASS: tests/Suite_ops/chord_creation_test_01.tdf
PASS: tests/Suite_ops/note_addition_test_01.tdf
PASS: tests/Suite_ops/phrase_creation_test_01.tdf
PASS: tests/Suite_ops/phrase_creation_test_02.tdf
PASS: tests/Suite_ops/raise_note_test_01.tdf
PASS: tests/Suite_ops/raise_octave_test_01.tdf
PASS: tests/Suite_ops/raise_octave_test_02.tdf
PASS: tests/Suite_ops/shift_phrase_test_01.tdf
PASS: tests/Suite_ops/shift_phrase_test_02.tdf
PASS: tests/Suite_ops/shift_phrase_test_03.tdf
PASS: tests/Suite_ops/type_promotion.tdf
PASS: tests/Suite_strings/print_everything.tdf
PASS: tests/Suite_strings/strings_test_01.tdf
PASS: tests/Suite_strings/strings_test_02.tdf
PASS: tests/Suite_strings/strings_test_03.tdf
PASS: tests/Suite_strings/strings_test_04.tdf

ALL 34 TESTS PASSED!
--------------------------
```

Figure 6.1: The output of test.py

Chapter 7

Lessons Learned

This section highlights some of the lessons that we have taken away as a group from collaborating on this project. Each group member wrote a brief paragraph of their own reflections on this project. They are presented here.

7.1 Matthew Duane

I learned that most successful modern languages are built on a finite set of basic elements that are then cast, abstracted, combined, and manipulated into a robust code base for development. At their core, virtually all Tonedef types are either ints or strings, yet with some abstraction and careful design we were able to create a language that was fairly customizable in creating audio output. Plus, we were able to provide some standard programming tenets like recursive functions and automatic type casting.

OCaml proved to be a cruel but fair taskmaster, beating me down with arcane error messages and a “unique” programming structure, but also providing the type of robust and efficient functionality that allowed us to create a language in about a thousand lines of code. I doubt more “popular” languages like Java or C can provide a fraction of this efficiency, and are just as idiosyncratic with respect to nomenclature.

Finally, I gained a greater appreciation for how programming languages function - how they progress from user-generated code to abstract syntax trees spawned from robust parsing, how the leaves of the AST are turned into system-usable bytecode, and how interpreters process that bytecode and ultimately perform the algorithms defined by the user. To have a hands-on, “soup-to-nuts” example of this process was enlightening, and the lessons learned here will undoubtedly affect my attempts to program more efficiently in the future.

7.2 Curtis Henkel

The biggest lesson I learned during this project was that when designing a language, every detail is important and deserves some thought and debate before moving onto implementation. Every design decision has repercussions on the complexity of the system. If you have been thorough when designing the language, then the process of implementing a compiler for it will go much more smoothly. Realizing an oversight in your initial design during implementation may be costly in terms of time and effort needed to rework the system.

My advice to future teams is this: Dont add unnecessary complexity. Pare down your ideas to their most basic and essential components. Pick a primary objective for what your language is trying to accomplish and focus in on that concept and what is necessary to provide that functionality. If you must have a broad list of objectives for your language, prioritize them and know which ones could be abandoned without destroying the system. Be flexible. Start early.

7.3 Kevin Ramkishun

This project taught me many lessons in organizing a large software project, documenting it, and working with others. This is not a project in which I could do most of the work myself; the work was truly split evenly and I had to learn to trust my team mates. The test suite helped with this; I knew that any changes made to the code base was valid because all of the tests that I had written were passed. I had never had an automated test suite such as the one used in this project, but from now on I will never have a software project without one.

The best part of working on the Tonedef compiler was the teamwork. Our team was extremely committed to producing quality code and being timely with our updates. Everyone was at the same level of understanding, and it seemed that there was *always* an update being commited. When some of us were busy with other classes and responsibilities, others would be actively working on the project, and vice-versa.

My advice to future teams is to get to know your team members well, because you will be spending a lot of time with them, both designing the language and then working through implementation issues. Find out everyones specific skills and faults and assign work based on what each member does best. Also, having a team captain to settle disputes is probably the best way when undertaking a project like this. Usually one person has the vision for the language, so let them make the final decision if team members see things differently.

7.4 Chatura Atapattu

> "Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away." - Antoine de Saint-Exupry

The hardest task in this project is not so much implementing the parser or compiler, but deciding what the language is and is not going to contain. At the beginning, Tonedef started off with numerous data types and structures, however as we worked on implementing it, we realized that everything in our language is easily represented using integers and character strings. This realization and simple approach, creating our language from the simplest building blocks and working our way up, made this project relatively enjoyable and hassle free. By understanding what your core goals are, and stripping away all the fat, its possible to come up with a clear set of concrete features for a language that is easily implementable.

A good team makes all the difference. Having worked on multiple teams in the past, and worked with three other teams during the course of this semester on large projects, I learned the difference having a good team makes. A good team is not necessarily the smartest team or the strongest one, a good team is when each of the members is dedicated, honest and forthcoming. It makes for good and clear communication, making working together stress free and a pleasant experience. When everyone is on the same page, and has the same goals, be it to do well in the project grade, or do the project well simply because they have a genuine interest in it, you'll find that the end result is much better and more rewarding. Often, our team worked on the project, enjoying it and seeing it as a nice break from all the other work we had to do. I've worked on a lot of projects in which I've had to do the grunt of the work or carry the team, but when you find a team that works equally because they want to, and not because they have to, you know youve found the right kind of people to work with.

Appendix A

Source Code Listing

This section lists all of the OCaml, Python, and Java source code that was written by members of our team. You may use the bookmarks on the side of the PDF to more easily navigate through this section. You may also use the table of contents to click on a section that you wish to peruse.

This section was built directly from the source code and accurately reflects the source code repository at the time of compiling this document.

A.1 test.py

```
import os, sys, subprocess as sub

num_errors = 0
num_failed = 0
num_warnings = 0
num_passed = 0

def run_tests():

    print '------------------------------'
    print 'Automatic Regression Tests'
    print '------------------------------'
    for root, dirs, files in os.walk('tests'):

        # ignore subversion directories
        if root.find('.svn') == -1:

            # test any file that ends in .tdf
            for file in files:
                if file.endswith('.tdf'):
                    filename = os.path.join(root, file)
                    test_file(filename)

    # Report results
    if num_errors == 0 and num_failed == 0 and num_warnings == 0:
        print '\nALL ' + str(num_passed) + ' TESTS PASSED!'
    else:
        print '\nThere are ' + str(num_failed) + ' failures,', \
            str(num_errors) + ' errors and', \
            str(num_warnings) + ' warnings.'

    print '------------------------------'

def test_file(filename):

    global num_errors, num_warnings, num_failed, num_passed

    infile = open(filename)

    # Add switch here to directly execute file
    # Pipe the input so we can analyze it
    proc = sub.Popen(['./tonedef'], stdin=infile, stdout=sub.PIPE, stderr=sub.PIPE)

    # Check if there was an error running tonedef
    errors = proc.stderr.readlines()
    if len(errors) > 0:
        print "ERROR: Running Tonedef failed on '" + filename + "', Output was:"
        print ''.join(errors)
        num_errors = num_errors + 1
        return

    gold_name = filename.replace('.tdf', '.gold')

    # make sure the gold file exists, open it, and compare.
    if not os.path.isfile(gold_name):
        print 'WARNING: The file "' + gold_name + '" does not exist. Skipping'
        num_warnings = num_warnings + 1
        return

    with open(gold_name, 'r') as gold:
        expected = gold.readlines()
        actual = proc.stdout.readlines()

        # workaround for windows files.. strip all '\r'
        expected = map(lambda x: x.replace('\r', ''), expected)
        actual = map(lambda x: x.replace('\r', ''), actual)

        if expected != actual:
            print 'FAIL: ' + filename
            num_failed = num_failed + 1

            # For debugging
            if len(sys.argv) > 1 and sys.argv[1] == '-v':
                print 'Expected:'
                print expected
                print ''
                print 'Actual:'
                print actual
                print ''

            return
    # If control gets here, we passed
    print 'PASS: ' + filename
    num_passed = num_passed + 1
    return 0

if __name__ == '__main__':

    # Make sure that everything is compiled first
    if not os.path.isfile('tonedef'):
        sub.Popen('make').wait()
    run_tests()
```

A.2 scanner.mll

```
{ open Parser

let process str =
  (* converts strings from program text to have appropriate escape characters *)
  let rec convert s =
    let len = String.length s in
    if (len < 2) then s else
      match (
        match (String.sub s 0 2) with
          | "\\n" -> "\n"
          | "\\t" -> "\t"
          | "\\r" -> "\r"
          | "\\\"" -> "\""
          | "\\\'" -> "\'"
          | "\\\\" -> "\\"
          | _ -> ""
      ) with
        | "" -> (String.sub s 0 1) ^ convert (String.sub s 1 (len - 1))
        | x -> x ^ convert (String.sub s 2 (len -2))
  in convert (String.sub str 1 (String.length str - 2))
}

rule token = parse
  | [' ' '\t' '\r' '\n'] { token lexbuf }
  | "/*" { comment lexbuf }
  | '+' { PLUS }
  | '-' { MINUS }
  | '*' { TIMES }
  | '/' { DIVIDE }
  | '%' { MOD }
  | "==" { EQ }
  | "!=" { NE }
  | '>' { GT }
  | '<' { LT }
  | "<=" { LTE }
  | ">=" { GTE }
  | '^' { CAROT }
  | "^^" { CAROT2 }
  | ':' { COLON }
  | "::" { COLON2 }
  | "**" { STAR2 }
  | "<<" { LEFT2 }
  | ">>" { RIGHT2 }
  | "@@" { AT2 }
  | "&&" { AND2 }
  | "||" { OR2 }
  | "//" { SLASH2 }
  | "\n" { NEWLINE }
  | "\r" { CARRETURN }
  | "\t" { TAB }
  | "\\" { BACKSLASH }
  | "\"" { DOUBLEQUOTE }
  | "if" { IF }
  | "else" { ELSE }
  | "for" { FOR }
  | "foreach" { FOREACH }
  | "in" { IN }
  | "while" { WHILE }
  | "function" { FUNCTION }
  | "return" { RETURN }
  | "true" { TRUE }
  | "false" { FALSE }
  | ',' { COMMA }
  | ';' { SEMICOLON }
  | '!' { EXCLAMATION }
  | '{' { OPENBRAC }
  | '}' { CLOSEBRAC }
  | '(' { OPENPAREN }
  | ')' { CLOSEPAREN }
  | '[' { OPENSEQBRAC }
  | ']' { CLOSESEQBRAC }
  | "int" { INT }
  | "string" { STRING }
  | "bool" { BOOLEAN }
  | "beat" { BEAT }
  | "pitch" { PITCH }
  | "sequence" { SEQUENCE }
  | "note" { NOTE }
  | "chord" { CHORD }
  | "phrase" { PHRASE }
  | "rhythm" { RHYTHM }
  | "void" { VOID }
  | ['0'-'9']+ as lxm { INTLIT(int_of_string lxm) }
  | '"' ([^'"']|"\\\"")* '"' as lxm { STRLIT (process lxm) }
  | '\'' ['1' '0' ' ' '-']* '\'' as lxm { RHYTHMLIT (process lxm) }
  | '$'['A'-'G' '_']['b' '#']?['0'-'9']? as lxm { PITCHLIT(lxm) }
  | ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { IDENT(lxm) }
  | '=' { ASSIGN }
  | eof { EOF }

and comment = parse
  | "*/" { token lexbuf }
  | _ { comment lexbuf }
```

A.3 parser.mly

```
%{ open Ast %}

%token PLUS MINUS TIMES DIVIDE MOD EOF EQ NE GT LT GTE LTE
%token NEWLINE DOUBLEQUOTE CARRETURN TAB BACKSLASH
%token CAROT CAROT2 COLON COLON2 STAR2 LEFT2 RIGHT2 AT2 SLASH2 AND2 OR2
%token IF ELSE FOR FOREACH IN WHILE FUNCTION RETURN PLAY PRINT MAIN
%token COMMA SEMICOLON EXCLAMATION TRUE FALSE
%token OPENBRAC CLOSEBRAC OPENPAREN CLOSEPAREN OPENSEQBRAC CLOSESEQBRAC
%token INT STRING BOOLEAN BEAT PITCH SEQUENCE NOTE CHORD PHRASE RHYTHM VOID
%token <int> INTLIT
%token <string> STRLIT
%token <string> PITCHLIT
%token <string> IDENT
%token <string> RHYTHMLIT
%token ASSIGN

%nonassoc NOELSE
%nonassoc ELSE

%left COMMA
%left ASSIGN
%left OR2
%left AND2
%left EQ NE
%left GT LT GTE LTE
%left PLUS MINUS
%left AT2
%left STAR2
%left LEFT2 RIGHT2 COLON2
%left COLON
%left CAROT CAROT2
%left TIMES DIVIDE MOD
%left SLASH2
%nonassoc UMINUS EXCLAMATION

%start program
%type <Ast.program> program

%%

td_type:
  | INT       { Int }
  | STRING    { String }
  | BOOLEAN   { Boolean }
  | BEAT      { Beat }
  | PITCH     { Pitch}
  | SEQUENCE  { Sequence }
  | NOTE      { Note }
  | CHORD     { Chord }
  | PHRASE    { Phrase }
  | RHYTHM    { Rhythm }
  | VOID      { Void }
```

```
typed_id:
  td_type IDENT { ($1, $2) }

expr:
  | expr PLUS expr                   { Binop($1, Add, $3) }
  | expr MINUS expr                  { Binop($1, Subtract, $3) }
  | expr TIMES expr                  { Binop($1, Multiply, $3) }
  | expr DIVIDE expr                 { Binop($1, Divide, $3) }
  | expr MOD expr                    { Binop($1, Mod, $3) }
  | expr EQ expr                     { Binop($1, Equal, $3) }
  | expr NE expr                     { Binop($1, NotEqual, $3) }
  | expr GT expr                     { Binop($1, GreaterThan, $3) }
  | expr LT expr                     { Binop($1, LessThan, $3) }
  | expr GTE expr                    { Binop($1, GtEqual, $3) }
  | expr LTE expr                    { Binop($1, LtEqual, $3) }
  | expr CAROT expr                  { Binop($1, RaiseNote, $3) }
  | expr CAROT2 expr                 { Binop($1, RaiseOctave, $3) }
  | expr COLON expr                  { Binop($1, BuildNote, $3) }
  | expr COLON2 expr                 { Binop($1, BuildChord, $3) }
  | expr LEFT2 expr                  { Binop($1, Apply, $3) }
  | expr RIGHT2 expr                 { Binop($1, ShiftPhrase, $3) }
  | expr STAR2 expr                  { Binop($1, CombinePhrases, $3) }
  | expr AT2 expr                    { Binop($1, AppendPhrases, $3) }
  | expr SLASH2 expr                 { Binop($1, BeatDivide, $3) }
  | expr AND2 expr                   { Binop($1, BoolAnd, $3) }
  | expr OR2 expr                    { Binop($1, BoolOr, $3) }
  | MINUS expr %prec UMINUS          { Unop(Negate, $2) }
  | EXCLAMATION expr                 { Unop(Invert, $2) }
  | PITCHLIT                         { Lit(PitchLit($1)) }
  | RHYTHMLIT                        { Lit(RhythmLit($1)) }
  | INTLIT                           { Lit(IntLit($1))}
  | STRLIT                           { Lit(StringLit($1)) }
  | TRUE                             { Lit(BoolLit(true)) }
  | FALSE                            { Lit(BoolLit(false)) }
  | IDENT OPENPAREN args CLOSEPAREN  { FnCall($1, List.rev $3) }
  | typed_id ASSIGN expr             { Assign($1, $3) }
  | IDENT ASSIGN expr                { Assign((Unspecified, $1), $3) }
  | IDENT                            { Id($1) }
  | OPENPAREN expr CLOSEPAREN        { $2 }
  | OPENSEQBRAC seq CLOSESEQBRAC     { Seq(List.rev $2) }
  | OPENSEQBRAC CLOSESEQBRAC         { Seq([]) }

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

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

stmt:
  | expr SEMICOLON
```

```
      { Execute($1) }
  | typed_id SEMICOLON
      { VarDecl($1) }
  | IF OPENPAREN expr CLOSEPAREN stmt ELSE stmt
      { IfThenElse($3, $5, $7) }
  | IF OPENPAREN expr CLOSEPAREN stmt %prec NOELSE
      { IfThenElse($3, $5, Block([])) }
  | FOR OPENPAREN expr SEMICOLON expr SEMICOLON expr CLOSEPAREN stmt
      { For($3, $5, $7, $9) }
  | FOREACH OPENPAREN typed_id IN expr CLOSEPAREN stmt
      { Foreach($3, $5, $7) }
  | WHILE OPENPAREN expr CLOSEPAREN stmt
      { While($3, $5) }
  | RETURN expr SEMICOLON
      { Return(Some($2)) }
  | RETURN SEMICOLON
      { Return(None) }
  | OPENBRAC stmts CLOSEBRAC
      { Block(List.rev $2) }

stmts:
  | { [] }
  | stmts stmt { $2 :: $1 }

func:
td_type FUNCTION IDENT OPENPAREN params CLOSEPAREN stmt { { return_type = $1;
    name = $3; paramlist = List.rev $5; body = $7 } }

params:
  | { [] }
  | typed_id                { $1 :: [] }
  | params COMMA typed_id   { $3 :: $1 }

funcs:
  | funcs func { $2 :: $1 }
  | { [] }

program:
  funcs EOF { List.rev $1 }
```

A.4 bytecode.mli

```
open Ast

type bytecode =
  (* push the int onto the stack *)
  | PushInt of int
  (* push a string onto the stack *)
  | PushString of string
  (* push a boolean literal onto stack *)
  | PushBool of bool
  (* push a beat tuple onto the stack *)
  | PushBeat
  | PushPitch of int
  | PushRhythm of string
  | PushEmpty of td_type
  (* Pop and discard an element from the stack *)
  | Pop
  (* 2 = Create a note from a pitch and beat on top of stack
   * 1 = Create a note from a pitch on top of the stack
   * 0 = Push the default note $_:1 onto the stack *)
  | CreateNote of int
  (* Create a chord on the top of the stack from a note and sequence *)
  | CreateChrd
  (* Create an empty sequence on top of stack *)
  | CreateSeq
  (* Create a phrase from a note and sequence on top of stack *)
  | CreatePhr
  (* Concatenates two stack elements into one string *)
  | Concat
  (* combines the top two stack elements and pushes the result back on the
   * stack one or both should be a type with a list representation: chord,
   * phrase, sequence *)
  | Combine
  (* If the top of the stack is a list type, remove the head and put the
   * corresponding type onto the stack. If the TOS is a note, put its pitch
   * and beat on the stack. If the TOS is a beat, put the integers numerator,
   * denominator on the stack *)
  | Decompose
  (* convert type of data on TOS from one type to another *)
  | ConvertType of td_type * td_type
  (* stack goes from pitch:int:TOP -> pitch:TOP with the second pitch
   * raised/lowered by the int from stack *)
  | AlterPitch of int
  (* Operations :
   * pop the top two items off the stack,
   * perform the operation,
   * places the result on top of the stack *)
  (* Add the top two items on the stack - x is in the name because we already
   * have an Add type label in AST *)
  | Addx
  (* Subtract the top two items on the stack *)
  | Sub
  (* Multiply the top two items on the stack *)
  | Mul
  (* Divide the top two items on the stack *)
  | Div
  (* Mod the top two items on the stack - x is in the name because we already
   * have a Mod type label in AST *)
  | Modx
  (* Logical AND *)
  | And
  (* Logical OR *)
  | Or
  (* Unary operations: only pop one operand off the stack, put result back on
   * top *)
  (* Negates an integer *)
```

```
  | Neg
  (* Logical inversion *)
  | Inv
  (* Comparisons *)
  (* one comparison opcode for all cases *)
  | Compare of bop
  (* checks if the TOP is an empty sequence,phrase, chord *)
  | IsEmpty
  (* Branching *)
  (* Jump to absolute address in bytecode program *)
  | Jmp of int
  (* Branch to address relative to PC *)
  | Bra of int
  (* Branch equal to zero/false - top of stack is zero - int is rel. to PC *)
  | Beq of int
  (* Branch not equal to zero/false - top of stack is not zer0 - int is
   * relative to PC*)
  | Bne of int
  (* Call a function *)
  | Call of int
  (* restore FP, SP, pop args, push result *)
  | Ret of int
  (* Memory management *)
  (* Load a variable onto stack from addr relative to fp *)
  | Load of int
  (* Store the top item on stack to addr relative to fp *)
  | Store of int
  (* push FP, SP -> FP , SP += i *)
  | Entry of int
  (* Terminate Program *)
  | Hlt
  (* can be used to put "comments" into bytecode *)
  | Nop of string

(* we don't have global variables so only need the bytecode instructions for
 * bytecode program *)
type bprogram = bytecode array

(* types for representing our data as memory in the bytecode interpreter *)
(* numerator, denominator *)
type mbeat = int * int
(* pitch, beat *)
type mnote = int * mbeat
(* list of notes *)
type mchord = mnote list
(* list of chordq *)
type mphrase = mchord list
type msequence = int list

(* integers, booleans, pitches, addresses are represented as int in our
 * memory structure *)
type memory =
  | 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

type stack = memory array
```

A.5 ast.mli

```
type bop = Add | Subtract | Multiply | Divide | Mod | GreaterThan | LessThan |
  Equal | NotEqual | GtEqual | LtEqual | RaiseNote | RaiseOctave | BuildNote |
  BuildChord | Apply | AppendPhrases | CombinePhrases | ShiftPhrase |
  BeatDivide | BoolAnd | BoolOr

type uop = Negate | Invert | Play | Print

type literal = IntLit of int | StringLit of string | PitchLit of string |
  BoolLit of bool | RhythmLit of string

type td_type = Int | String | Boolean | Pitch | Note | Beat | Sequence |
  Chord | Phrase | Rhythm | Unspecified | Void

type typed_id = td_type * string

type esc_char = Doublequote | Newline | Tab | Carreturn | Backslash

type expr =
  | Unop of uop * expr
  | Binop of expr * bop * expr
  | Lit of literal
  | FnCall of string * expr list
  | Assign of typed_id * expr
  | Seq of expr list
  | Id of string

type stmt =
  | VarDecl of typed_id
  | IfThenElse of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | Foreach of typed_id * expr * stmt
  | While of expr * stmt
  | Return of expr option
  | Execute of expr
  | Block of stmt list
  | Empty

type func = {
  return_type : td_type;
  name : string;
  paramlist : typed_id list;
  body : stmt;
}

type program = func list
```

A.6 compiler.ml

```
open Ast
open Bytecode
open Helper

module StringHash = Hashtbl.Make
  (struct
    type t = string (* type of keys *)
    let equal x y = x = y (* use structural comparison *)
    let hash = Hashtbl.hash (* generic hash function *)
   end)

(* used for translating expressions
 * each expression translates to a list of bytecode
 * and a td_type = the type of the result of the expression which will be on
 * TOS after executing this code *)
type typed_code = {
  typ : td_type;
  code: bytecode list;
}

(* combine_code: typed_code list -> bytecode list *)
let combine_code tclist =
  List.fold_right (fun a b-> (a.code @ b)) tclist []

(* string -> bprogram -> () *)
let write_bp filename bprog =
  let ochannel = open_out filename in
  output_value ochannel bprog;
  close_out ochannel

(* string -> bprogram -> () *)
let write_bp_text filename bprog =
  let ochannel = open_out filename in
  (Array.iter (fun bcode -> output_string ochannel (
    (Helper.string_of_bc bcode)^"\n")
   ) bprog;
   close_out ochannel)

(* enum : int ->   a list -> (int *   a ) list *)
let rec enum stride n = function
  | [] -> []
  | hd::tl -> (n, hd) :: enum stride (n+stride) tl

(* string_map_pairs:StringMap   a -> (int *   a ) list -> StringMap   a *)
let string_map_pairs map pairs =
    List.fold_left (fun m (i, n) -> StringMap.add n i m) map pairs

(* that the AST representation of a tonedef program and translate into a
 * bytecode program representation *)
(* translate: Ast.program -> Bytecode.bprogram *)
let translate (func_list) =
  let built_in_functions = ["print"; "play"] in
  (* Goes through all the functions and builds a map of their function
   * signature as a td_type list with the first element the return_type
   * of the func and the rest of the elements the ordered types of the
   * paramlist *)
  (* type: string => td_type list *)
  let sig_map =
    StringMap.add "print" [Void; String] (
      StringMap.add "play" [Void; Phrase] (
        Helper.get_signature_map func_list)) in
  (* Reads in all of the functions in the program and stores them in
   * func_map *)
  (* function name (string) maps to labels (int) *)
  let func_map =
    let start_map = string_map_pairs StringMap.empty (enum (-1) (-1)
                                                   built_in_functions) in
    string_map_pairs start_map (enum 1 1 (List.map (fun f-> f.name)
                                              func_list)) in
  (* Translate a function into its bytecode *)
  (* fdecl : Ast.func
   * translate_fn : Ast.func -> Bytecode.bytecode list *)
  let rec translate_fn fdecl =
    let localslist = Helper.checkfunction sig_map fdecl in
    (* s the type checking/scoping checks for fdecl *)
    let num_locals = List.length localslist
    (* number of local variables throughout f to be placed on
     * top of stack *)
    and num_formals = List.length (fdecl.paramlist)
    and local_offsets = enum 1 1 localslist
    and formal_offsets = enum (-1) (-2)
      (List.map snd fdecl.paramlist) in
    (* var_map is a mapping of variable names to their location
     * on the stack relative to the frame pointer *)
    let var_map = string_map_pairs StringMap.empty
      (local_offsets @ formal_offsets)
    and type_hash = StringHash.create (num_formals+num_locals) in
    (* put params and types into the type_hash - locals will be
     * added as met in program; this is because a name can be one
     * type inside one loop and another type in a different loop *)
    let _ =
      List.map (fun (t,id) -> StringHash.add type_hash id t) fdecl.paramlist in
    (* function that returns a bytecode list containing the opcode for store
     * it gets the offset for that var from var_map
     * store_var: string -> bytecode list *)
    (* set_type: string -> td_type -> () *)
    let set_type var typ = StringHash.add type_hash var typ
    (* get_type: string -> td_type *)
    and get_type var = try (StringHash.find type_hash var)
      with Not_found -> raise (Failure ("Variable name has no type:" ^ var))
    and get_func_sig fname = try (StringMap.find fname sig_map)
      with Not_found -> raise (Failure ("Undefined Function Signature: "^fname))
    and get_func_addr fname = try StringMap.find fname func_map
      with Not_found -> raise (Failure ("Undefined Function:" ^ fname))
    and get_var_addr id = try StringMap.find id var_map
      with Not_found -> raise (Failure ("Undefined variable: " ^ id))
    and conversion tc exp_t = if (tc.typ = exp_t) then tc
      else if (Helper.is_promotable_to tc.typ exp_t) then {
        typ = exp_t;
        code = tc.code @ [ConvertType (tc.typ,exp_t)]
      } else
        raise(Failure ("Type error - cannot convert types " ^
                        (Printer.string_of_type_id tc.typ) ^" to " ^
                        (Printer.string_of_type_id exp_t))) in
    (* converts an expression into a typed_code
     * (td_type+bytecode list) with recursive calls as needed *)
    let rec translate_type_expr = function
    Lit(StringLit(s)) -> { typ = String; code = [Bytecode.PushString (s)] }
      | Lit (IntLit (i)) -> { typ = Int; code = [Bytecode.PushInt (i)] }
      | Lit (BoolLit(b)) -> { typ = Boolean; code = [Bytecode.PushBool(b)] }
      | Lit (PitchLit(s)) -> { typ = Pitch; code =
          [PushPitch(Helper.int_of_pitchstring(s))] }
      | Lit (RhythmLit(r)) -> { typ = Rhythm; code = [PushRhythm (r)] }
      | Id (id) -> { typ = (get_type id); code =
          [Bytecode.Load (get_var_addr id)] }
      | Assign ((t, id), e) ->
        let _ = match t with
          | Unspecified -> ()
          | _ -> set_type id t (* set the type in type_hash *)
        and e_tc = translate_type_expr e
        and id_typ = get_type id in
        let etc_converted = conversion e_tc id_typ in {
          typ = id_typ;
          code = etc_converted.code @ [Store (get_var_addr id)]
        }
      | FnCall(fname, elist) ->
        let expected_types = get_func_sig fname
        and tc_list = (List.map translate_type_expr (List.rev elist)) in
        let return_type = List.hd expected_types
        and arg_types = List.tl expected_types
        and fn_address = get_func_addr fname in
        let tc_list_converted =
          List.map2 conversion tc_list (List.rev arg_types) in
        {
          typ = return_type;
          code = (combine_code tc_list_converted) @ [Jmp(fn_address)]
        }
      | Unop (op, e1) ->
        let tc1 = translate_type_expr e1 in
        let (result_type, opcode) =
          match (op, tc1.typ) with
            (* Push a Neg command on stack along with int value *)
            | (Negate, Int) -> (Int, [Neg])
            (* Push a neg command on stack along with beat *)
            | (Negate, Beat) -> (Beat, [Neg])
            | (Invert, Boolean) -> (Boolean, [Inv])
            | _ -> raise (Failure "Type error - unary operator") in
        { typ = result_type;
          code = tc1.code @ opcode
        }
      | Binop (e1, op, e2) ->
        let tc1 = translate_type_expr e1
        and tc2 = translate_type_expr e2 in
        (* checks that the operand types are allowed for this op *)
        let (ta,tb) = Helper.binop_check_operand_types op tc1.typ tc2.typ in
        (* addes conversion of types if needed to match an allowed operand type
         * of the operator *)
        let tc1 = conversion tc1 ta
        and tc2 = conversion tc2 tb in
        let (result_type, opcode) =
          match (op, tc1.typ, tc2.typ) with
            | (Add, Int, Int) -> (Int, [Addx])
            (* Addition operator for beats *)
            | (Add, Beat, Beat) -> (Beat, [Addx])
            | (Add, String, String) -> (String, [Concat])
            | (Add, Chord, Chord) -> (Chord, [Concat])
            | (Add, Rhythm, Rhythm) -> (Rhythm, [Concat])
            | (Add, Sequence, Sequence) -> (Sequence, [Concat])
            | (Subtract, Int, Int) -> (Int, [Sub])
            | (Subtract, Beat, Beat) -> (Beat, [Sub])
            | ((GreaterThan | LessThan | GtEqual | LtEqual | Equal | NotEqual),
               (Int | Beat | Pitch | Boolean | Note), (Int | Beat | Pitch |
                   Boolean | Note)) -> (Boolean, [Compare(op)])
            (* Multiplication operator for ints *)
            | (Multiply, Beat, Beat) -> (Beat, [Mul])
            (* Multiplication operator for ints *)
            | (Multiply, Int, Int) -> (Int, [Mul])
            (* Multiplication operator for ints *)
            | (Divide, Int, Int) -> (Int, [Div])
            | (Divide, Beat, Beat) -> (Beat, [Div])
            | (Mod, Int, Int) -> (Int, [Modx])
            (* Pushes a beat onto the stack *)
            | (BeatDivide, Int, Int) -> (Beat, [PushBeat])
            | (BuildNote, (Note|Pitch), Beat) -> (Note, [CreateNote(2)])
            | (RaiseNote, Note , Int) -> (Note, [AlterPitch(1)])
            | (RaiseNote, Pitch, Int) -> (Pitch, [AlterPitch(1)])
            | (RaiseOctave, Note, Int) -> (Note, [AlterPitch(12)])
            | (RaiseOctave, Pitch, Int) -> (Pitch, [AlterPitch(12)])
            | (BoolAnd, Boolean, Boolean) -> (Boolean, [And])
            | (BoolOr, Boolean, Boolean) -> (Boolean, [Or])
            | (BuildChord, Note, Sequence) -> (Chord, [CreateChrd])
            | (Apply, Note, Sequence) -> (Phrase, [CreatePhr])
            | (Apply, Phrase, Rhythm) -> (Phrase, [CreatePhr])
            | (AppendPhrases, Phrase, Phrase) -> (Phrase, [Concat])
            | (ShiftPhrase, Phrase, Beat) -> (Phrase, [CreatePhr])
            | (CombinePhrases, Phrase, Phrase) -> (Phrase, [Combine])
            | _ -> (Void, [Nop("Binop catchall case")]) in
        { typ = result_type;
          code = tc1.code @ tc2.code @ opcode }
      | Seq (elist) ->
        let tc_list = List.map translate_type_expr elist in
        let tc_list_converted =
          List.map (fun tc -> (conversion tc Int)) tc_list in
        let codelist = List.fold_left (fun c tc -> tc.code @ (Combine :: c))
          []  tc_list_converted in
        { typ = Sequence; code = PushEmpty(Sequence) ::codelist }
    (* translate the expression e and returns only the code, not the type *)
    and translate_expr e = let tc = translate_type_expr e in
                           tc.code
    (* converts a statement to bytecode with recursive calls when necessary *)
    and translate_stmt = function
    (* Reads a block of statements and translates them each in order *)
    Block(stmtlist) -> List.concat (List.map translate_stmt stmtlist)
      | VarDecl(t ,id) ->
        (* set the type in type_hash *)
        let _ = set_type id t in
        (* put default value on stack and store it *)
        (match t with
          | Int -> [PushInt 0]
          | Pitch -> [PushPitch 0]
          | Boolean -> [PushBool false]
          | String -> [Bytecode.PushString("")]
          | Note -> [CreateNote(0)]
          | x -> [PushEmpty x]
        ) @ [Store (get_var_addr id)]
      | Execute(e) -> translate_expr e @ [Bytecode.Pop]
      | IfThenElse (p, tstmt, fstmt) -> let tstmt' = translate_stmt tstmt
      and fstmt' = translate_stmt fstmt
      and p' = conversion (translate_type_expr p) Boolean in
                                        p'.code @ [Bytecode.Beq(2 + List.length tstmt')] @
                                          tstmt' @ [Bytecode.Bra(1 + List.length fstmt')] @
      | While (p, body) -> let p' = conversion (translate_type_expr p) Boolean
      and body' = translate_stmt body in
                           [Bytecode.Bra (1+ List.length body')] @ body' @ p'.code @
                             [Bytecode.Bne (-(List.length body' + List.length p'.code))]
      | Return (e_opt) ->
        let expected_type = List.hd (get_func_sig fdecl.name) in
        let translation = match e_opt with
          | Some(e) ->
            let tc = conversion (translate_type_expr e) expected_type in
            tc.code
          | None ->
            if (expected_type = Void) then
              (* need to push something, because need a return value on top *)
              [PushInt (0)]
            else raise (Failure "Non-void functions must return a value") in
        translation @ [Bytecode.Ret(num_formals)]
      | For (e1, e2, e3, b) -> let e1' = Execute e1
      and e3' = Execute e3 in
                               translate_stmt (Block([e1'; While(e2, Block([b; e3']))]))
      | Foreach ((t,id), exp , body) ->
```

```
        let _ = set_type id t in
        let exp_bc = translate_expr exp
        and body_bc = translate_stmt body in
        let offset = List.length body_bc + 4 in
        exp_bc @ [Bra (offset); Decompose; Store(get_var_addr id); Pop]
        @ body_bc @ [IsEmpty; Beq (-offset); Pop]
      | _ -> [] in
    [Bytecode.Entry(num_locals)] @ (translate_stmt fdecl.body)
    @ [Bytecode.PushInt(0); Bytecode.Ret(num_formals)] in

  (* Code executed to start the program: Jsr main; halt *)
  let entry_function = try
                        [Jmp (StringMap.find "main" func_map); Hlt]
    with Not_found -> raise (Failure ("no \"main\" function")) in
  (* Compile the functions *)
  let func_bodies = entry_function :: List.map translate_fn func_list in

  let (fun_offset_list , _) = List.fold_left (fun (l,i) f ->
    (i :: l,(i+List.length f))) ([],0)
    func_bodies in
  let func_offset = Array.of_list (List.rev fun_offset_list) in
  let bytecodeprogram = Array.of_list (List.map
                                        (function
                                        Bytecode.Jmp(i) when i > 0 -> Bytecode.Jmp (func_
                                          | _ as s-> s) (List.concat func_bodies)) in
  bytecodeprogram
```

A.7 execute.ml

```
open Ast
open Bytecode
open Helper

let rec gcd a = function 0 -> abs(a) | b -> gcd b (a mod b)
let reduce_beat (n,d) =
  let g = gcd n d in
  (n / g, d / g)

(* Convert a mbeat to a jfugue duration *)
let rec mbeat_to_jfugue b =
  let (n,d) = b in
  if (n < 0) then ""
  else if (n mod d == 0) then
    String.make (n / d) 'w'
  else if (n >= d) then
    "w" ^ mbeat_to_jfugue (reduce_beat (n - d, d))
  else if ((2 * n) >= d) then
    "h" ^ mbeat_to_jfugue (reduce_beat ((2 * n) - d, 2 * d))
  else if ((4 * n) >= d) then
    "q" ^ mbeat_to_jfugue (reduce_beat ((4 * n) - d, 4 * d))
  else if ((8 * n) >= d) then
    "i" ^ mbeat_to_jfugue (reduce_beat ((8 * n) - d, 8 * d))
  else if ((16 * n) >= d) then
```

```
    "s" ^ mbeat_to_jfugue (reduce_beat ((16 * n) - d, 16 * d))
  else ""

(* total_sixteenths: string -> int
   counts the number of sixteenths in the string of w,h,q,i,s chars *)
let rec total_sixteenths s =
  let len = String.length s in
  if (len = 0) then 0
  else
    let c = String.get s 0
    and rest = String.sub s 1 (len - 1) in
    (match c with
      | 'w' -> 16
      | 'h' -> 8
      | 'q' -> 4
      | 'i' -> 2
      | 's'-> 1
      | _ -> 0)
    + (total_sixteenths rest)

(* Convert a mnote to a jfugue pitch and duration *)
(* second element of tuple is the # of sixteenths in this duration *)
let mnote_to_jfugue n =
  let p = fst n and b = snd n in
  let convo_b = mbeat_to_jfugue b
  and convo_p =
    if (p = 0) then "R"
    else
      let o = (p-1) / 12
      and p_partial = match (p mod 12) with
        | 0 -> "B"
        | 1 -> "C"
        | 2 -> "C#"
        | 3 -> "D"
        | 4 -> "D#"
        | 5 -> "E"
        | 6 -> "F"
        | 7 -> "F#"
        | 8 -> "G"
        | 9 -> "G#"
        | 10 -> "A"
        | 11 -> "A#"
        (* Will never happen but match must be exhaustive *)
        | _ -> ""  in
      p_partial ^ (string_of_int o) in
  (convo_p ^ convo_b, total_sixteenths convo_b)

(* Convert a mchord to a concatenated list of jfugue notes *)
(* second element in the return tuple is the largest number of sixteenths of
 * any note in the chord *)
let mchord_to_jfugue c =
  let listc = List.map mnote_to_jfugue c in
  let rec find_max max = function
    | hd::tl -> find_max (if (hd>max) then hd else max) tl
    | [] -> max in
  let max_num = find_max 0 (List.map snd listc) in
  let stringc = String.concat "+" (List.map fst listc) in
  (stringc, max_num)

(* Convert a mchord to a concatenated list of jfugue notes or a rest
 * duration. Second in the return tuple is the number of sixteenth
 * notes that this chord lasts *)
let mchord_to_jfugue_wrapper c =
  if List.length c > 0 then
    mchord_to_jfugue c
  else
    ("Rs", 1)

(* Convert a mphrase to a jfugue pattern *)
let mphrase_to_jfugue p =
  fst (List.fold_left (fun (s,n) e -> if (n > 1) then (s, n-1)
    else let (str, num) = mchord_to_jfugue_wrapper e in
         (s ^ " " ^ str, num)) ("",0) p)

(* Play phrase using jfugue *)
let jfugue_string = "java -cp jfugue.jar: PlayJFugue"
(* If java throws an error about no MIDI Synthesizers being available
 * use the following line instead of the one above to attempt to use
 * PulseAudio *)
(* let jfugue_string = "padsp java -cp jfugue.jar: PlayJFugue" *)
let play_jfugue phrase = Unix.system (jfugue_string ^ " " ^
                                        (mphrase_to_jfugue phrase))

(* play phrase *)
let play_phrase = function
  | MemPhrase(p) -> ignore(play_jfugue p)
  | _ -> ignore("")

(* memory -> string *)
let rec mem_to_string = function
  | MemInteger(i) -> string_of_int (i)
  | MemString(s) -> s
  | MemBool(b) -> string_of_bool b
  | MemPitch(i) -> "Pitch:"^(string_of_int i)
  | MemBeat(n,d) -> (string_of_int(n)) ^ "/" ^ (string_of_int(d))
  | MemSequence(ilist) -> "["^ (String.concat ","
                                 (List.map string_of_int ilist)) ^ "]"
  | MemNote(p,(n,d)) -> fst (mnote_to_jfugue (p,(n,d)))
  (* "Note:" ^ (string_of_int p) ^ "," ^ (string_of_int n) ^ "/" ^
   * (string_of_int d) *)
  | MemChord(c) -> fst (mchord_to_jfugue c)
  (* "Chord [" ^ mchrd_to_string(c) ^ "]" *)
  | MemPhrase(p) -> mphrase_to_jfugue p
  (* "Phrase {" ^ mphr_to_string p ^ "}" *)
  | MemRhythm(r) -> r
and mchrd_to_string = function
hd :: tl -> (mem_to_string (MemNote(hd))) ^ (mchrd_to_string tl)
  | _ -> ""
```

```
and mphr_to_string = function
hd :: tl -> (mem_to_string (MemChord(hd))) ^ (mphr_to_string tl)
  | _ -> ""

let rec make_empty x =
  if (x <=0) then []
  else [] :: (make_empty (x-1))
and chord_to_phrase chd =
  let num_sixteenths = List.map (fun (p, (n,d)) -> (16 * n / d)) chd in
  let rec find_max max vals =
    match vals with
      | hd::tl -> find_max (if (hd > max) then hd else max) tl
      | [] -> max in
  let max = find_max 0 num_sixteenths in
  chd :: (make_empty (max-1))

(* read_bc : string -> Bytecode.bprogram *)
let read_bc filename =
  let ichannel = open_in filename in
  let bp = input_value ichannel in
  close_in ichannel; bp

(* show values in memory stack *)
let show_stack st =
  let print_stack entry = print_string(mem_to_string(entry) ^ " ") in
  Array.iter print_stack st

(* run: Bytecode.bprogram -> () *)
let run prog =
  let stack = Array.make 1024 (MemInteger(0)) in
  let get2 i = (stack.(i-2), stack.(i-1))
  and get1 i = stack.(i-1) in
  let rec gcd a = function 0 -> abs(a) | b -> gcd b (a mod b) in
  let rec exec fp sp pc =
    (* prints pc, sp, sp, bytecode before executing the bytecode *)
    (*
     * let _ = List.map print_string (List.map (fun x-> string_of_int x ^
     *    " ") [pc; sp; fp]) in
     * let _ = print_endline (string_of_bc (prog.(pc))) in
     *)
    match prog.(pc) with
      | PushInt(i) -> stack.(sp) <- MemInteger(i); exec fp (sp + 1) (pc + 1)
      | PushString(s) -> stack.(sp) <- MemString(s); exec fp (sp + 1) (pc + 1)
      | PushBool(b) -> stack.(sp) <- MemBool(b); exec fp (sp + 1) (pc + 1)
      | PushPitch(p) -> stack.(sp) <- MemPitch(p); exec fp (sp + 1) (pc + 1)
      | PushRhythm(r) -> stack.(sp) <- MemRhythm(r); exec fp (sp + 1) (pc + 1)
      | Load(offset) -> stack.(sp) <- stack.(fp + offset);
        exec fp (sp + 1) (pc + 1)
      | Store(offset) -> stack.(fp + offset) <- stack.(sp - 1);
        exec fp sp (pc + 1)
      | Jmp(-1) -> print_string (mem_to_string stack.(sp - 1));
        exec fp sp (pc + 1)
      | Jmp(-2) -> ignore(play_phrase stack.(sp - 1)); exec fp sp (pc + 1)
      | Jmp(address) -> stack.(sp) <- MemInteger(pc + 1);
        exec fp (sp + 1) address
      | Ret(num) ->
        let new_fp = int_value_of_mem (stack.(fp))
        and new_pc = int_value_of_mem (stack.(fp - 1))
        and new_sp = (fp-num) in
        (* put return value on top of stack *)
        stack.(new_sp - 1) <- stack.(sp - 1) ;
        exec new_fp new_sp new_pc
      | Entry(num) -> stack.(sp) <- MemInteger(fp);
        exec sp (sp + num + 1) (pc + 1)
      | Concat ->
        stack.(sp - 2) <-
          (match (get2 sp) with
            | (MemString (s1), MemString(s2)) -> MemString(s1^s2)
            | (MemChord(c1), MemChord(c2)) -> MemChord(c1 @ c2)
            | (MemPhrase(p1), MemPhrase(p2)) -> MemPhrase(p1 @ p2)
            | (MemSequence(s1), MemSequence(s2))-> MemSequence(s1 @ s2)
            | (MemRhythm(r1), MemRhythm(r2)) -> MemRhythm(r1 ^ r2)
            | _ -> raise (Failure "Type error - bad operand types for concat "));
        exec fp (sp - 1) (pc + 1)
      | Addx ->
        (match (get2 sp) with
          | (MemInteger(i1), MemInteger(i2)) -> stack.(sp - 2) <- MemInteger(i1+i2)
          | (MemBeat(i1), MemBeat(i2)) -> let n = ((fst(i1) * snd(i2)) + (fst(i2) *
                                                                      snd(i1))) and 
                                            let gcd_div = gcd n d in
                                            (* Factors out the denominator with the gcd *)
                                            stack.(sp - 2) <- MemBeat(n/gcd_div, d/gcd_div)
          | _ ->  raise (Failure "Type error - bad operand types for + "));
        exec fp (sp - 1) (pc + 1)
      | Sub ->
        (match (get2 sp) with
          | (MemInteger(i1), MemInteger(i2)) -> stack.(sp - 2) <- MemInteger(i1-i2)
          | (MemBeat(i1), MemBeat(i2)) -> let n = ((fst(i1) * snd(i2)) - (fst(i2) *
                                                                      snd(i1))) and 
                                            let gcd_div = gcd n d in
                                            (* Factors out the denominator with the gcd *)
                                            stack.(sp - 2) <- MemBeat(n / gcd_div, d / gcd_d
          | _ -> raise (Failure "Type error - bad operand types for - "));
        exec fp (sp - 1) (pc + 1)
      | And ->
        (match (get2 sp) with
          | (MemBool(b1), MemBool(b2)) -> stack.(sp - 2) <- MemBool(b1 && b2)
          | _ -> raise (Failure "Type error - bad operand types for &&"));
        exec fp (sp - 1)(pc + 1)
      | Or ->
        (match (get2 sp) with
          | (MemBool(b1), MemBool(b2)) -> stack.(sp - 2) <- MemBool(b1 || b2)
          | _ -> raise (Failure "Type error - bad operand types for ||"));
        exec fp (sp - 1)(pc + 1)
      (* Multiplication bytecode *)
      | Mul ->
        (match (get2 sp) with
          | (MemInteger(i1), MemInteger(i2)) -> stack.(sp - 2) <- MemInteger(i1*i2)
```

```
        | (MemBeat(i1), MemBeat(i2)) -> let n = (fst(i1) * fst(i2))
        and d = (snd(i1) * snd(i2)) in
                                          let gcd_div = gcd n d in
                                          (* Factors out the denominator with the gcd *)
                                          stack.(sp - 2) <- MemBeat((n/gcd_div, d/gcd_div)
        | _ -> raise (Failure "Type error - bad operand types for * "));
      exec fp (sp - 1) (pc + 1)
    | Div -> (* Division bytecode *)
      (match (get2 sp) with
        | (MemInteger(i1), MemInteger(i2)) ->
          if (i2 = 0) then
            raise (Failure "Value error - cannot divide int by 0 or null")
          else
            stack.(sp - 2) <- MemInteger(i1/i2)
        | (MemBeat(i1), MemBeat(i2)) ->
          (* check to see if denominator of i1 or numerator of i2 is 0,
           * because that is an error *)
          if ((snd(i1) = 0) or (fst(i2) = 0)) then
            raise (Failure "Value error - cannot divide beat by 0 or null")
          else
            let n = (fst(i1) * snd(i2)) and d = (snd(i1) * fst(i2)) in
            let gcd_div = gcd n d in
            (* Factors out the denominator with the gcd *)
            stack.(sp - 2) <- MemBeat((n/gcd_div, d/gcd_div));
        | _ -> raise (Failure "Type error - bad operand types for * "));
      exec fp (sp - 1) (pc + 1)
    (* Modulo bytecode *)
    | Modx ->
      (match (get2 sp) with
        | (MemInteger(i1), MemInteger(i2)) ->
          if (i2 = 0) then
            raise (Failure "Value error - cannot modulo int by 0 or null")
          else
            stack.(sp - 2) <- MemInteger(i1 mod i2)
        | _ -> raise (Failure "Type error - bad operand types for % "));
      exec fp (sp - 1) (pc + 1)
    | PushBeat ->
      (match (get2 sp) with
        | (MemInteger(i1), MemInteger(i2)) ->
          if (i2 = 0) then
            raise (Failure " Value error - 0 beat value invalid ")
          else
            (* Make beat by combining the two ints pushed onto stack *)
            stack.(sp - 2) <- (MemBeat(i1, i2));
        | _ -> raise (Failure "Type error - beat components must be integers"));
      exec fp (sp - 1) (pc + 1)
    | Neg ->
      (match (get1 sp) with
        (* Negate the top value on the stack and store it to the stack *)
        | (MemInteger(i1)) -> stack.(sp - 1) <- MemInteger(-1 * i1)
        | (MemBeat(i1)) -> stack.(sp - 1) <- MemBeat(-1 * fst(i1), snd(i1))
        | _ -> raise (Failure "Type error - bad operand for negation"));
      exec fp sp (pc + 1)
    | Inv ->
      (match (get1 sp) with
        (* Invert the bool value at the top of the stack and store result *)
        | (MemBool(i1)) -> stack.(sp - 1) <- MemBool(not i1)
        | _ -> raise (Failure "Type error - Logical inversion only available \
    for boolean"));
      exec fp sp (pc + 1)
    | Compare (cop) ->
      let (i1,i2) = match (get2 sp) with
        | (MemInteger(a), MemInteger(b)) -> (a,b)
        | (MemBeat(a,c), MemBeat(b,d)) -> (a*d,b*c)
        | (MemPitch(a), MemPitch(b)) -> (a,b) (* Compares pitches *)
        | (MemBool(a), MemBool(b)) -> (convert_bool(a), convert_bool(b))
        (* This will need to be streamlined *)
        | (MemNote(a, (b,c)), MemNote(d, (e,f))) -> (convert_bool(a = d),
                                                     convert_bool(b * f = c * e))
        | _ -> raise (Failure "Cannot compare types") in
      stack.(sp - 2) <- MemBool
        (match cop with
          | GreaterThan -> i1 > i2
          | LessThan -> i1 < i2
          | GtEqual -> i1 >= i2
          | LtEqual -> i1 <= i2
          | Equal -> i1 = i2
          | NotEqual -> i1 != i2
          | _ -> raise (Failure "Compiler error - improper comparison type"));
      exec fp (sp - 1) (pc + 1)
    | CreateNote(i) ->
      let default_beat = (1,1) and default_pitch = 0 in
      stack.(sp-i) <-
        (match i with
          | 2 -> (match (get2 sp) with
              | (MemPitch(p), MemBeat(b)) -> MemNote(p,b)
              | _ -> raise (Failure "Improper types for note contruction"))
          | 1 -> (match (get1 sp) with
              | MemPitch (p) -> MemNote(p, default_beat)
              | _ -> raise (Failure "Improper types for note contruction"))
          | _ -> MemNote(default_pitch, default_beat));
      exec fp (sp-i+1) (pc + 1)
    | PushEmpty(t) ->
      stack.(sp) <-
        (match t with
          | Sequence -> MemSequence([])
          | Chord -> MemChord([])
          | Phrase -> MemPhrase ([])
          | _ -> MemInteger(0))
      ; exec fp (sp + 1) (pc + 1)
    | CreateSeq ->
      stack.(sp) <- MemSequence([]); exec fp (sp + 1) (pc + 1)
    | CreateChrd ->
      (match (get2 sp) with
        (* Create a chord from a note and a sequence applied to note *)
        | (MemNote(n), MemSequence(s)) ->
          (* Duplicate the note the number of times as the length of
           * the sequence *)
          let rec createnotelist list note iter =
            if iter == List.length s then list
            else
              createnotelist (note :: list) note (iter + 1) in
          let noteslist = createnotelist [] (n) 0 in
          (* Apply sequence to notes list *)
          let apply_sequence = List.map2 (fun n amt ->
            ((fst n) + amt, snd n)) noteslist s in
          stack.(sp - 2) <- MemChord(apply_sequence)
        | _ -> raise (Failure "Improper Types for Creating Chord."));
      exec fp (sp - 1) (pc + 1)
    | CreatePhr ->
      (match (get2 sp) with
        | (MemNote(n), MemSequence(s)) ->
          let (p, (num,den)) = n in
          (* Duplicate the note as many times as the number of ints in
           *the sequence *)
          let make_note_list offset =
            let p2 = p+offset in
            let p2 = if (p2<0) then 0 else p2 in
            [ (p2, (num,den))] :: (make_empty ((16*num/den) -1)) in
          let apply_sequence = List.concat (List.map make_note_list s) in

          (* Create a phrase from the list of chords *)
          stack.(sp - 2) <- MemPhrase(apply_sequence)
        | (MemPhrase(p), MemBeat(n,d)) ->
          stack.(sp - 2) <- MemPhrase((make_empty (16 * n / d)) @ p)
        | (MemPhrase(p), MemRhythm(s)) ->
          let rec pop_first = function
            | [] :: tl -> pop_first tl
            | hd :: tl -> (hd,tl)
            | [] -> ([],[]) in
          let rec expand_section (n, s) =
            if (String.length s > 0) then
              (String.sub s 0 1) ^ (String.make (n-1) '-') ^ (expand_section
                                                            (n, String.sub s 1 (Strin
            else "" in
          let rec expand_str s =
            if (s = "") then "" else
              let i = try (String.index s ' ') with Not_found -> (String.length s) in
              (if (i >= 16) then expand_section (1,String.sub s 0 16)
               else if (i >= 8) then expand_section (2, String.sub s 0 8)
               else if (i >= 4) then expand_section (4, String.sub s 0 4)
               else if (i >= 2) then expand_section (8, String.sub s 0 2)
               else if (i >= 1) then expand_section (16, String.sub s 0 1)
               else "") ^
                try (expand_str (String.sub s (i+1) (String.length s -(i+1))))
                with Invalid_argument(_) -> "" in
          let rec condense_str_rec s (tf, n) =
            if (String.length s > 0) then
              let rest = String.sub s 1 (String.length s -1) in
              match (String.get s 0) with
                | '1' -> (tf, n) :: (condense_str_rec rest (true, 1))
                | '0' -> (tf, n) :: (condense_str_rec rest (false,1))
                | _ -> condense_str_rec rest (tf, n+1)
            else [ (tf, n) ]
          and condense_str s = condense_str_rec s (false, 0) in
          let rec mapphrase old_phr tuple_list =
            match tuple_list with
              | (_, 0)::tl -> mapphrase old_phr tl
              | (false, n)::tl -> []::(mapphrase old_phr ((false, n-1)::tl))
              | (true, n)::tl ->
                let (h, rest_phr) = pop_first old_phr in
                let new_chd = List.map (fun (pit,(num,den)) -> (pit,(n,16))) h in
                new_chd :: (mapphrase rest_phr ((false, n-1)::tl))
              | _ -> [] in
          let new_mphrase = mapphrase p (condense_str (expand_str s)) in
          stack.(sp - 2) <- MemPhrase(new_mphrase)
        | _ -> raise (Failure "Improper types for creating phrase")
      );
      exec fp (sp - 1) (pc + 1)

    (*stack.(sp) <- MemPhrase([]); exec fp (sp + 1) (pc + 1) *)
    | Combine -> stack.(sp - 2) <- (match (get2 sp) with
        | (MemSequence(slist), MemInteger(i)) -> MemSequence(i::slist)
        | (MemInteger(i), MemSequence(slist))-> MemSequence(i::slist)
        | (MemNote(n), MemChord(c)) -> MemChord (n::c)
        | (MemChord(c), MemNote(n))-> MemChord (n::c)
        | (MemPhrase(p1), MemPhrase(p2)) ->
          let rec combine_one_spot = function
          (h1 :: t1 , h2 :: t2) -> (List.concat [h1; h2]) ::
            (combine_one_spot (t1, t2))
            | (x, []) -> x
            | ([] , x) -> x in
          MemPhrase (combine_one_spot (p1, p2))
        | _ -> raise (Failure "Type error - combine op"));
      exec fp (sp - 1) (pc + 1)
    | IsEmpty ->
      (*print_string (mem_to_string (get1 sp));*)
      stack.(sp) <- MemBool
        (match(get1 sp) with
          | (MemSequence([])|MemPhrase([])|MemChord([])) -> true
          | _ -> false);
      exec fp (sp + 1) (pc + 1)
    | Decompose ->
      let (x,y) =
        (match (get1 sp) with
          | MemSequence(hd::slist) -> (MemSequence(slist), MemInteger(hd))
          | MemPhrase(hd::clist) -> (MemPhrase(clist), MemChord(hd))
          | MemChord(hd::nlist) -> (MemChord(nlist), MemNote(hd))
          | MemNote(p,b) -> (MemPitch(p), MemBeat(b))
          | MemBeat(n,d) -> (MemInteger(n), MemInteger(d))
          | _ ->
            raise (Failure ("Decompose on improper type " ^
                              (mem_to_string (get1 sp))))) in
      stack.(sp - 1)<- x; stack.(sp) <-y;
      exec fp (sp + 1) (pc + 1)
    | AlterPitch(factor) ->
```

```
      let new_p p i = let np = p+i*factor in if (np<0 || p=0) then 0 else np in
      stack.(sp - 2) <-
        (match (get2 sp) with
          |  (MemNote(p, b), MemInteger(i)) -> MemNote(new_p p i, b)
          | (MemPitch(p), MemInteger(i)) -> MemPitch(new_p p i)
          | _ -> raise (Failure "Type error - AlterPitch"));
      exec fp (sp - 1) (pc + 1)
    | ConvertType (t1, t2) ->
      (* Unnecessary but redundant check *)
      let allow_promote = is_promotable_to t1 t2 in
      if (allow_promote) then
        let d1 = get1 sp in
        stack.(sp - 1) <-
          (match (t1, t2) with
            | (Int, Boolean) ->
              (* int value != 0 means true, else false *)
              if (int_value_of_mem(d1) = 0) then MemBool(false) else MemBool(true)
            | (Boolean, Int) ->
              (* True value is 1, else 0 *)
              if (d1 = MemBool(true)) then MemInteger(1) else MemInteger(0)
            | (Int, Beat) ->
              (* the value of beat is rounded down to an integer *)
              MemBeat(int_value_of_mem(d1), 1)
            | (Beat, Int) -> (match d1 with
                | MemBeat(n,d) -> MemInteger(n/d)
                (* beats are rational numbers so the value of the int is
                 * preserved as a beat *)
                | _ -> raise (Failure "Type error - Beat to Int failed"))
            (* pitch is promoted to a note with duration = 0 *)
            | (Pitch, Note) -> MemNote(int_value_of_mem(d1), (0, 1))
            | (Beat, Note) -> (match d1 with
                | MemBeat(n,d) -> MemNote(0, (n,d))
                (* beat is promoted to a note with that duration and pitch =
                 * where $_ = 0 *)
                | _ -> raise (Failure "Type error - Beat to Note failed"))
            (*note is demoted to only its pitch value *)
            | (Note, Pitch) -> (match d1 with
                | MemNote(p, b) -> MemPitch(p)
                (* For the case when the pitch is being raised or lowered octaves,
                 * this test catches it. *)
                | MemPitch(p) -> stack.(sp - 1)
                | _ -> raise (Failure "Type error - Note to Pitch failed"))
            (*note is demoted to only its beat value *)
            | (Note, Beat) -> (match d1 with
                | MemNote(p, b) -> MemBeat(b)
                | _ -> raise (Failure "Type error - Note to Beat failed"))
            (* note is promoted to a chord contain just that one note *)
            | (Note, Chord) -> (match d1 with
                | MemNote(n) -> MemChord(n :: [])
                | _ -> raise (Failure "Type error - Note to Chord failed"))
            | (Note, Phrase) -> (match d1 with
                | MemNote(n) -> MemPhrase (chord_to_phrase [n])
                | _ -> raise (Failure "Type error - Note to Phrase failed"))
            (* chord is promoted to a phrase contain just that one chord *)
            | (Chord, Phrase) -> (match d1 with
                | MemChord(c) -> MemPhrase(chord_to_phrase c)
                | _ -> raise (Failure "Type error - Chord to Phrase failed"))
            | (_, String) -> MemString(mem_to_string d1)
            | _ -> raise (Failure "Type error - Cannot be promoted"));
        (* print_endline("sp = " ^ string_of_int(sp)); show_stack stack;*)
        exec fp (sp) (pc + 1);
    | Hlt -> ()
    (* Branch to forward i number of instructions *)
    | Bra(i) -> exec fp sp (pc+i)
    | Bne(i) ->
      exec fp (sp - 1) (pc + (let adx =
                              if (int_value_of_mem(stack.(sp - 1)) != 0)
                              then i else 1 in adx))
    | Beq(i) -> exec fp (sp - 1) (pc + (let adx =
                                        if (int_value_of_mem(stack.(sp - 1)) = 0)
                                        then i  else 1  in adx))
    | Pop -> exec fp (sp - 1) (pc + 1)
    (* non-implemented cases are just nop's to start *)
    | _ -> exec fp sp (pc + 1) in
  exec 0 0 0
```

A.8 helper.ml

```
open Ast
open Str
open Bytecode

module StringMap = Map.Make(String)
module StringHash = Hashtbl.Make
  (struct
    (* type of keys *)
    type t = string
    (* use structural comparison *)
    let equal x y = x = y
    (* generic hash function *)
    let hash = Hashtbl.hash
   end)

(* called to check if types are allowed in a foreach construction *)
let validate_foreach_types t1 t2 = match (t1, t2) with
  | (Int, Sequence) -> true
  | (Note, Chord) -> true
  | (Chord, Phrase) -> true
  | _ -> false

let are_equiv t1 t2 =
  let check a b = match (a,b) with
    (* (String, Boolean) -> true *)
    | (Int, Boolean) -> true
    | (Int, Beat) -> true
    | (Note, Pitch) -> true
    | (Note, Beat) -> true
    | (x, y) -> x = y in
  (check t1 t2) or (check t2 t1)

let promote = function
  | Int -> String
  | Pitch -> Note
  | Note -> Chord
  | Chord -> Phrase
  | Phrase -> String
  | Beat -> String
  | Sequence -> String
  | Boolean -> String
  | Rhythm -> String
  | _ -> Void

let rec is_promotable_to t1 t2 = match (t1,t2) with
  | (Void,Void) -> true
  | (Void,_) -> false
  | _ -> (are_equiv t1 t2) or (is_promotable_to (promote t1) t2)

(* checks that the two types t1 t2 are valid types for the operator op,
 * returns a 3-tuple (operand_type_a, operand_type_b, result_type)
 * where operand_type_a,b may be different than t1,t2 because of type
 * conversion *)
(* fun: Ast.bop -> Ast.td_type -> Ast.td_type -> (td_type, td_type, td_type) *)
let binop_check_operands op t1 t2 =
  let list_of_allowed_types = match op with
    | Add -> [(Chord, Chord, Chord);(Beat, Beat, Beat); (Int, Int, Int);
              (String, String, String); (Rhythm, Rhythm, Rhythm);
              (Sequence, Sequence, Sequence)]
    | Subtract -> [(Beat, Beat, Beat); (Int, Int, Int)]
    | Multiply -> [(Beat, Beat, Beat); (Int, Int, Int)]
    | Divide -> [(Beat, Beat, Beat);(Int, Int, Int)]
    | Mod -> [(Int, Int, Int)]
    | BeatDivide -> [(Int, Int, Beat)]
    | BuildNote -> [(Pitch, Beat, Note)]
    | RaiseNote -> [(Note, Int, Note); (Pitch, Int, Pitch)]
    | RaiseOctave -> [(Note, Int, Note); (Pitch, Int, Pitch)]
    | BoolAnd -> [(Boolean, Boolean, Boolean)]
    | BoolOr -> [(Boolean, Boolean,Boolean)]
    | BuildChord -> [(Note, Sequence, Chord)]
    | Apply -> [(Note, Sequence, Phrase); (Phrase, Rhythm, Phrase)]
    | AppendPhrases -> [(Phrase, Phrase, Phrase)]
    | ShiftPhrase -> [(Phrase, Beat, Phrase)]
    | CombinePhrases -> [(Phrase,Phrase, Phrase)]
    | (GreaterThan | LessThan | GtEqual | LtEqual) -> [(Beat, Beat, Boolean);
                                                  (Int, Int, Boolean); (Pitch, Pitch,
    | (Equal | NotEqual) -> [(Beat, Beat, Boolean);
                             (Boolean, Boolean, Boolean); (Int, Int, Boolean);
                             (Pitch,Pitch, Boolean); (Note,Note, Boolean)] in
  let rec find_first_match = function
    | (ta, tb, tc) :: tl -> if (t1 = ta && t2 = tb) then Some(ta,tb, tc)
      else find_first_match tl
    | _ -> None
  and find_first_promotable_match = function
    | (ta,tb, tc) :: tl -> if ((is_promotable_to t1 ta) &&
                               (is_promotable_to t2 tb)) then Some(ta,tb, tc)
      else find_first_promotable_match tl
    | _ -> None in
  match (find_first_match list_of_allowed_types) with
    | Some(ta,tb, tc) -> (ta,tb, tc)
    | None ->
      match (find_first_promotable_match list_of_allowed_types) with
        | Some(ta,tb, tc) -> (ta,tb, tc)
        | None -> raise (Failure ("Bad operand types for operator " ^
                                  (Printer.string_of_bop op)))

let binop_check_operand_types op t1 t2 =
  let (ta, tb, _) = binop_check_operands op t1 t2 in
  (ta,tb)

let binop_check_operand_return op t1 t2 =
  let (_, _, tc) = binop_check_operands op t1 t2 in
  tc

(* checkfunction:
 * signatures: a map from strings(function names) to td_type lists
 * (returntype::paramtypeslist for each function)
 * f : Ast.func
 * returns a list of local variable names used throughout the function
 *)
let checkfunction signatures f =
  let idhash = StringHash.create 10 in
  let scope = StringMap.add "return" f.return_type StringMap.empty in
  let scope = List.fold_left (fun m (t, id) -> StringMap.add id t m)
    scope f.paramlist in
  let addid scope id t = if (StringMap.mem id scope)
    then raise (Failure ("Id (" ^ id ^ ") already declared in this scope"))
    else (StringHash.add idhash id id; StringMap.add id t scope) in
  (* checkstmt: scope -> stmt -> scope *)
  let rec checkstmt scope statement =
    match statement with
      | VarDecl ((t,id)) -> addid scope id t
      | IfThenElse(e,a,b) -> let newscope = (checkexpr scope e) in
                             let predtype = (checktype newscope e) in
                             if (are_equiv predtype Boolean)
                             then (ignore(checkstmt newscope a); ignore(checkstmt newscope
                                   scope)
                             else raise (Failure ("Bad type of predicate in if statement"))
      | For(e1,e2,e3,s) -> let newscope = (checkexpr scope e1) in
                           ignore(checktype newscope e1) ; ignore(checktype newscope e2);
                           ignore(checktype newscope e3); ignore(checkstmt newscope s) ; sc
      | Foreach ((t,id), e, s) -> let newscope = addid scope id t
      and enumtype = checktype scope e in
                                  if (validate_foreach_types t enumtype) then
                                    (ignore(checkstmt newscope s); scope)
                                  else scope
      | While (e, s) -> let newscope = (checkexpr scope e) in
```

```
                        let predtype = (checktype newscope e) in
                        if (are_equiv predtype Boolean) then (ignore(checkstmt newscope s)
                                                                    scope)
                        else raise (Failure ("Bad type of predicate in while statement"))
        | Return (e_opt) ->
          let returntype = match e_opt with
            | Some(e) -> checktype scope e
            | None -> Void
          and expectedtype = StringMap.find "return" scope in
          if (is_promotable_to returntype expectedtype) then scope
          else raise (Failure ("Returning wrong type in " ^ f.name))
        | Execute (e) -> let newscope = checkexpr scope e in newscope
        | Block (slist) -> List.fold_left checkstmt scope slist
        | _ -> scope
    (* checkexpr: scope -> stmt -> scope *)
    (* takes a scope and adds a new string-> type mapping if the expression is
     * a typed assignment *)
    and checkexpr scope expression =
      match expression with
        | Assign ((Unspecified,id), e) -> ignore(checktype scope expression);
          scope
        | Assign ((t, id), e) ->
          let typ = checktype scope e in
          if (is_promotable_to typ t) then addid scope id t
          else raise (Failure "Right-side of assignment not equivalent to \
      assignment type")
        | _ -> ignore(checktype scope expression) ; scope
    (* checkexpr: scope -> expr -> td_type *)
    and checktype scope expression = match expression with
      | Unop (op, e) -> checktype scope e
      | Binop (e1, op, e2) -> let t1 = checktype scope e1
      and t2 = checktype scope e2 in ignore(t2);
                              binop_check_operand_return op t1 t2
      | Lit (x) -> (match x with
          | IntLit(_) -> Int
          | StringLit(_)->String
          | PitchLit(_)->Pitch
          | BoolLit(_) -> Boolean
          | RhythmLit(_) -> Rhythm)
      | FnCall (calledf, elist) ->
        ignore (List.map (checktype scope) elist);
        let fsig = StringMap.find calledf signatures in
        let arglist = List.tl fsig in
        if ((List.length arglist) = (List.length elist))
        then List.hd fsig
        else raise (Failure ("Incorrect number of arguments for function: " ^
                              calledf))
      | Assign ((t, id), e) -> if (StringMap.mem id scope) then
          let exp_typ = checktype scope e
          and var_typ = StringMap.find id scope in
          if (is_promotable_to exp_typ var_typ) then var_typ
          else raise (Failure "Right-side of assignment not equivalent to assignment \
      type")
        else (match t with
          | Unspecified -> raise (Failure ("Assigning to unscoped variable:" ^ id))
          | _ -> raise (Failure ("Declaring variable type improperly - must be at \
      beginning of statement")))
      | Seq (elist) ->
        ignore (List.map (checktype scope) elist);
        Sequence
      | Id (id) -> if (StringMap.mem id scope) then StringMap.find id scope
        else raise (Failure ("Unscoped Variable:" ^ id)) in
    ignore(checkstmt scope f.body);
    StringHash.fold (fun k a b -> a::b) idhash []

  let get_signature_map flist=
    let get_types_list f =
      f.return_type :: (List.map fst f.paramlist) in
    List.fold_left (fun m f -> StringMap.add f.name (get_types_list f) m)
      StringMap.empty flist

  (* checks that all variables used are in scope and types of expressions,
   * funciton arguments are corrects *)
  let check_all flist =
    List.map (checkfunction (get_signature_map flist)) flist

  let string_of_bc = function
    | PushInt(i) -> "PushInt " ^ (string_of_int i)
    | PushString(s) -> "PushString " ^ (String.escaped s)
    | PushBool(b) -> "PushBool " ^ (string_of_bool b)
    | PushBeat -> "PushBeat "
    | PushRhythm(r) -> "PushRhythm " ^ r
    | PushPitch(p) -> "PushPitch " ^ (string_of_int p)
    | PushEmpty (t) -> "PushEmpty " ^ (Printer.string_of_type_id t)
    | Pop -> "Pop"
    | CreateNote(i) -> "CreateNote "^ (string_of_int i)
    | CreateChrd -> "CreateChrd"
    | CreateSeq -> "CreateSeq"
    | CreatePhr -> "CreatePhr"
    | Combine -> "Combine"
    | Concat -> "Concat"
    | ConvertType(a,b) -> "ConvertType " ^ (Printer.string_of_type_id a) ^ "->" ^
      (Printer.string_of_type_id b)
    | Compare (binop) -> "Compare " ^ (Printer.string_of_bop binop)
    | Decompose -> "Decompose"
    | Addx -> "Addx"
    | Sub -> "Sub"
    | Mul -> "Mul"
    | Div -> "Div"
    | And -> "And"
    | Or -> "Or"
    | Neg -> "Neg"
    | Inv -> "Inv"
    | Jmp(i) -> "Jmp " ^ (string_of_int i)
    | Bra(i) -> "Bra " ^ (string_of_int i)
    | Beq(i) -> "Beq " ^ (string_of_int i)
    | Bne(i) -> "Bne " ^ (string_of_int i)
    | Call(i) -> "Call" ^ (string_of_int i)
    | Ret(i) -> "Ret " ^ (string_of_int i)
    (* Memory management *)
    | Load(i) -> "Load " ^ (string_of_int i)
    | Store(i) -> "Store " ^ (string_of_int i)
    | Entry(i) -> "Entry " ^ (string_of_int i)
    | Hlt -> "Hlt"
    | Nop (s) -> "Nop " ^ (String.escaped s)
    | IsEmpty -> "IsEmpty"
    | AlterPitch(f) -> "AlterPitch " ^ (string_of_int f)
    | _ -> "Nop"

  let print_bp bprog =
    let counter = ref 0 in
    Array.iter (fun x -> (print_endline (string_of_int(counter.contents) ^ " " ^
                                          string_of_bc x); incr counter)) bprog

  let int_value_of_mem = function
    | MemInteger(i) -> i
    | MemBool(b) -> if b then 1 else 0;
    | MemPitch(p) -> p
    | _ -> raise (Failure ("Compile error: memory not an int "))

  let convert_bool a = (match a with
    | true -> 1
    | false -> 0)

  let int_of_pitchstring ps =
    let len = String.length ps in
    let octave = String.sub ps (len-1) 1
    and p = String.sub ps 1 (len-2) in
    let conv_o = try (int_of_string octave) with Failure(_) -> 0
    and conv_p = match p with
      | ("C"|"B#") -> 1
      | ("C#"|"Db") -> 2
      | ("D") -> 3
      | ("D#"|"Eb") -> 4
      | ("E"|"Fb") -> 5
      | ("F"|"E#") -> 6
      | ("F#"|"Gb") -> 7
      | ("G") -> 8
      | ("G#"|"Ab") -> 9
      | ("A") -> 10
      | ("A#"|"Bb") -> 11
      | ("B"|"Cb") -> 12
      | _ -> 0 in
    (12 * conv_o) + conv_p
```

A.9 tonedef.ml

```
open Ast
open Helper
open Execute
```

```
let print = false

type action = PrintBytecode | Execute | PrintCode

let _ =
  let action =
    if Array.length Sys.argv > 1 then
      List.assoc Sys.argv.(1) [("-b", PrintBytecode);
              ("-e", Execute);
              ("-p", PrintCode)]
    else
      Execute in
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  match action with
    | PrintBytecode -> Helper.print_bp (Compiler.translate program)
    | Execute -> Execute.run (Compiler.translate program)
    | PrintCode -> let listing = Printer.string_of_program program in
       print_string listing
```

A.10 printer.ml

```
open Ast

let string_of_uop o =
  match o with
    | Negate -> "-"
    |Invert -> "!"
    | Play -> "play"
    | Print -> "print"

let string_of_lit l =
  match l with
    | IntLit(l) -> string_of_int l
    | StringLit(l) -> l
    | PitchLit(l) -> l
    | BoolLit(b) -> string_of_bool b
    | RhythmLit(r) -> r

let string_of_bop o =
  match o with
    | Add -> "+"
    | Subtract -> "-"
    | Multiply -> "*"
    | Divide -> "/"
    | Mod -> "%"
    | Equal -> " == "
    | NotEqual -> " != "
    | LessThan -> " < "
    | LtEqual -> " <= "
    | GreaterThan -> " > "
    | GtEqual -> " >= "
    | RaiseNote -> " ^ "
    | RaiseOctave -> " ^^ "
    | BuildNote -> " : "
    | BuildChord -> " :: "
    | Apply -> " << "
    | AppendPhrases -> " @@ "
    | CombinePhrases -> " ** "
    | ShiftPhrase -> " >> "
    | BeatDivide -> " // "
    | BoolAnd -> "&&"
    | BoolOr -> "||"

let string_of_type_id v =
  match v with
    | Int -> "int "
    | String -> "string "
    | Boolean -> "bool "
    | Pitch -> "pitch "
    | Note -> "note "
    | Beat -> "beat "
    | Sequence -> "sequence "
    | Chord -> "chord "
    | Phrase -> "phrase "
    | Rhythm -> "rhythm "
    | Unspecified -> " "
    | Void -> "void "

let string_of_typed_id (v, e) = string_of_type_id v ^ e

let rec string_of_expr = function
  | Unop(o, e1) -> string_of_uop o ^ " " ^ string_of_expr e1
  | Id(s) -> s
  | Lit(l) -> string_of_lit l
  | Binop(e1, o, e2) ->
    string_of_expr e1 ^ " " ^
      string_of_bop o ^ " " ^
      string_of_expr e2
  | Assign(v, e) -> (string_of_typed_id v ) ^ " = " ^ string_of_expr e
  | Seq( list ) -> "[ " ^
    (String.concat ", " (List.map string_of_expr list)) ^ " ]"
  | FnCall(f, el) ->
    f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"

let rec string_of_stmt = function
  |Block(stmts) ->
    "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | VarDecl(v, e) -> (string_of_type_id v) ^ e ^ ";\n"
  | IfThenElse(e, s1, s2) -> "if (" ^ string_of_expr(e) ^ ")\n " ^
    string_of_stmt(s1) ^ "\n else " ^ 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
  | Foreach(v, e, s) -> "foreach (" ^ (string_of_typed_id v) ^ " in " ^
      string_of_expr e ^ " ) " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Return(Some(expr)) -> "return " ^ string_of_expr expr ^ ";\n"
  | Return(None) -> "return;"
  | Execute(expr) -> string_of_expr expr ^ ";\n"
  | Empty -> "Empty;\n"

let string_of_func func =
  (* Return type *)
  "\n" ^ string_of_type_id func.return_type ^
    (* Function name *)
    "" ^ func.name ^ "(" ^
    (* Param list *)
    String.concat ", " (List.map string_of_typed_id func.paramlist) ^
    (* Body *)
    ")\n" ^ string_of_stmt func.body

let string_of_program funcs =
  String.concat "\n" (List.map string_of_func funcs)
```

A.11 PlayJFugue.java

```
import org.jfugue.*;

public class PlayJFugue {
        public static void main(String[] args) {
                if (args.length == 0) {
                        System.out.println("No JFugue string provided!");
                        System.exit(1);
                }

                String patternString = "";

                for (int i = 0; i < args.length; i++) {
                patternString += args[i] + " ";
                }

                Player player = new Player();
                Pattern pattern = new Pattern(patternString.trim());
                player.play(pattern);
        }
}
```

Appendix B

Sample Tonedef Programs

This section includes some of the sample Tonedef programs that have been compiled and run successfully.

B.1 quicksort.tdf

This program comes up with a random sequence of notes, plays the notes, sorts them using a recursive quicksort algorithm, and then plays the sorted notes.

```
void function main()
{

        sequence random =        choose_random ( 1);  /* 1-5 for argument */

        note n = $F5: 1//16;
        phrase a = n << random;
        sequence sorted = quicksort(random);
        phrase b = n << sorted;

        phrase c = a @@ (b>>1);

        play(c);
}

/* sorts a sequence of integers */
sequence function quicksort ( sequence arr ) {

        int count = 0;
        sequence lower = [];
        sequence higher = [];
        int p = 0;
        foreach ( int x in arr ){
                if ( count == 0) {
                        p = x;  /* choose partition */
                } else {
                        /* partition sequence into two */
                        if ( x <=p )
                                lower = lower + [ x ];
                        else higher = higher + [x];
                }
                count = count + 1;
        }
        if ( count == 0 )  return [] ;
        else return quicksort(lower) + [p] + quicksort(higher);

        return [];

}

/* returns one of 5 sequences of 100 random integeters between -50 and 50
        arg x: int [1-5]  */
sequence function choose_random ( int x ) {

        if ( x == 1 )
                return [-26, -43, 45, 33, -5, -18, 30, -40, -23, 26, -50, 21, -10, 39, -24
        else if (x == 2)
                return [ 26, 41, -31, -37, -1, -46, 10, 11, 33, 13, -31, -5, -48, 20, 12,
        else if (x == 3)
                return [6, 46, -29, 40, 17, 11, -44, -7, 36, -3, -3, 15, 3, 17, 0, -11, -6
        else if (x == 4)
                return [-9, -19, -15, -20, -38, 40, -31, 9, -46, -30, 42, 5, 28, -34, 25,
        else if (x == 5)
                return [-3, 10, -32, -24, -24, -6, 12, -33, -16, -40, -13, -41, -15, -7, -
        return [];
}
```

B.2 brownian.tdf

This program implements a random walk of notes. Each subsequent note either stays at the same pitch, goes up by 1 pitch, or decreases by 1 pitch.

```
void function main()
{
     int seed = 300;
     int num = 400;
     sequence maj1 = [ 0 , 4, 7, 12 ];
     sequence maj2 = [ 0 , 2, 4, 5, 7, 9, 11 ];

     note c = $C0: 1//16;
     sequence allmaj = [];

     foreach ( int x in [ 1, 2, 3, 4, 5, 6, 7, 8] ){
        foreach ( int y in maj2)
            allmaj = allmaj + [ 8 * x + y ];
     }

   int idx = 64;
   sequence brown = [];
     foreach ( int r in first (  get_rand( seed), num)  ){
          idx = idx + r;
          brown = brown + [idx];
```

```
    }

   brown = news ( allmaj , brown);
   /*print (brown);*/
   play ( c << brown );
}

sequence function get_rand ( int s){

   sequence r = [1, -1, -1, 1, -1, 1, 0, -1, -1, 1, 1, 0, -1, -1, 0, 0, 0, 1, 1, 0, -1, -1
 1, 1, 1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1, -1, 0, 0, -1, 1, 0, 0, 0, -1, 0, 0, -1, -1, 0,
-1, 1, 0, -1, -1, -1, 1, -1, 0, -1, 1, 0, -1, 0, -1, 0, 0, -1, -1, -1, 0, 1, -1, -1, 0, -1
0, -1, 1, -1, 1, 1, -1, 0, 0, 0, 1, 0, 1, -1, -1, 0, -1, 0, 1, 0, 1, -1, 0, 1, 0, 1, 0, 1,
 0, 1, -1, 0, 0, 1, 1, 1, -1, 1, 1, 1, -1, 1, -1, 0, -1, 1, -1, 1, 1, -1, -1, -1, -1, -1,
, 1, 1, -1, -1, 0, 1, 1, 0, 1, -1, -1, -1, 1, -1, 1, -1, 0, 0, 1, 1, -1, -1, 0, 0, -1, 1,
-1, 0, 0, 1, -1, 1, -1, -1, 1, 0, -1, 1, 1, -1, 0, -1, 1, 0, 0, 1, -1, -1, 1, -1, 0, 0, -1
1, 0, -1, 1, 1, 0, 0, -1, 1, 1, 1, -1, 0, 1, 1, 1, 1, 0, 0, -1, 0, -1, 1, 1, -1, 0, 1, -1,
 0, -1, 1, -1, -1, 1, 0, -1, 0, 1, 0, 0, 1, 0, -1, -1, -1, 1, 1, 0, 1, -1, 0, 1, 0, 0, 1,
 -1, 1, -1, 1, 0, 0, 1, -1, 0, 1, 1, 0, 1, -1, 1, 1, -1, 1, 0, -1, 1, 0, 0, 0, 0, -1, -1,
, -1, 0, -1, -1, -1, 0, -1, -1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 0, -1, 0, 0, 1, 1, -1, 1
1, -1, 0, -1, -1, -1, -1, 1, 0, -1, -1, -1, -1, 1, -1, -1, 1, 0, 1, 0, 1, -1, 0, 0, 0, 0,
1, 0, 1, -1, 1, -1, 0, 0, -1, -1, 0, -1, -1, 0, -1, 0, 1, -1, 1, -1, 0, -1, 0, 1, -1, 1, 0
 0, -1, -1, 0, 1, -1, -1, -1, 1, 0, 0, 0, 1, 1, -1, 0, 1, 0, 1, 0, 0, -1, -1, 1, 0, 1, 0,
1, -1, 0, 0, 1, 0, 1, 0, -1, -1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, -1, 1, 1, 0,
 -1, 1, -1, 1, -1, 1, 0, -1, -1, -1, -1, 0, 0, 1, 0, 0, -1, -1, 1, 1, -1, -1, -1, -1, -1,
, 1, -1, 0, -1, 0, 0, 0, 0, 0, 0, 1, 1, 0, -1, -1, -1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
-1, -1, 0, -1, -1, -1, -1, 0, 0, 1, -1, 1, -1, 1, -1, -1, 1, 0, -1, 0, 1, -1, -1, -1, -1,
0, 1, 0, 1, -1, -1, -1, 0, 0, -1, 1, 0, -1, 1, 0, 0, 1, 1, 1, 0, -1, 0, 1, -1, 1, 1, 1, -1
 0, 0, -1, 1, 1, 0, 1, -1, -1, -1, 1, 1, 1, 0, -1, 0, -1, 1, 0, 0, 1, -1, 1, 1, 1, 0, 0, 0
, 1, -1, 1, 0, 1, 1, 1, -1, 0, 1, -1, 0, 0, 1, -1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, -1, 1,
 0, 1, 0, -1, -1, -1, 0, 1, 0, 0, -1, 1, 0, 0, 0, 0, -1, 0, 1, -1, -1, -1, -1, 1, 0, 0, -1
 1, 1, -1, 1, -1, -1, 0, -1, 1, 0, -1, 0, 0, -1, 1, 1, 0, 1, 0, -1, 0, -1, 1, -1, -1, -1,
 0, 1, 1, 1, -1, -1, 1, 0, 1, -1, 0, 1, -1, -1, 0, 1, -1, 1, -1, 0, -1, -1, 1, 1, 0, 0, 0,
 1, 0, 0, 0, 1, -1, -1, 0, 1, 1, -1, 1, 1, -1, -1, 0, 0, 1, -1, -1, 0, -1, -1, 1, 0, -1, -
 -1, -1, -1, -1, 0, 0, 1, -1, 0, -1, 1, 1, -1, 1, -1, -1, -1, 0, 0, 1, -1, -1, -1, 0, -1,
, 0, 0, 1, -1, -1, -1, 0, 0, 0, 0, 1, 0, -1, 0, 0, -1, -1, 1, 0, -1, 0, -1, 1, 1, 0, -1, 1
0, -1, 1, 1, 0, -1, -1, -1, 0, 0, 1, -1, 1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1, 0, -1, -1
, -1, -1, -1, -1, 1, 0, 1, 1, 1, 1, -1, -1, 1, 0, -1, 1, 0, 1, 0, 1, -1, -1, 1, 0, -1, -1,
, 1, 0, 0, -1, 1, 1, 0, -1, 0, 1, -1, 0, 0, 0, 0, 1, 0, 1, 0, -1, 0, 1, 0, -1, -1, 1, 0, 0
1, 1, -1, -1, 1, 1, -1, 1, -1, 0, 1, 1, 0, 0, 1, 1, -1, -1, 1, 0, 1, 0, -1, 0, 0, 1, 0, 0,
 -1, 1, 1, 0, 0, 0, 0, -1, 0, -1, -1, 1];

   sequence a = [];
   sequence b  =[];
   int count = 0;
   foreach ( int i in r){
     if ( count < s ) {
        a = [i] + a;
     } else b = [i] + b;
     count = count + 1;
   }
   return  rev(b) + rev(a)  ;
```

```
}
sequence function first ( sequence s, int num ) {
   int count = 0;
   sequence r = [];
   foreach ( int i in s) {
     if (count < num ){
        r = r + [i];
     }
     count = count + 1;
   }
   return r;
}

sequence function rev (sequence s ){
   sequence r =  [];
   foreach (int i in s){

      r = [i] + r;
   }
   return r;
}

sequence function news (sequence ns, sequence is ) {
      sequence r = [];
      foreach ( int i in is ) {
                 if ( i < 1 ) { i = 1; }
                 int count = 1;
                 foreach ( int n in ns ) {
                   if (count == i ) {
                      r = r + [ n ];
                   }
                   count = count + 1;
                 }

      }
   return r;

}
```

B.3 demo_intervals.tdf

This program demonstrates the overlapping of two notes in a chord.

```
void function main()
{
     sequence maj1 = [ 0 , 4, 7, 12 ];
     sequence maj2 = [ 0 , 2, 4, 5, 7, 9, 11, 12 ];

     sequence s1 = news ( maj2 , [ 1, 3, 5, 8 ] );
     sequence s2 = news ( maj2,  [ 1, 8, 1, 8] );
     sequence s3 = news ( maj2,  [ 1, 3, 1, 3] );
     sequence s4 = news ( maj2,  [ 1, 5, 1, 5] );
     sequence s5 = news ( maj2,  [ 1, 2, 3, 4] );
```

```
      sequence s6 = news ( maj2,  [ 5, 6, 7, 8] );

      chord cmaj = $C5 :: maj1;
      chord gmaj = $G5 :: maj1;


     sequence y = (s3 + s3 + s5 + s5);
     print (s3);
     sequence A1 = (s3 + s3 + s4 + s4 + s2 + s2 );
     sequence A2 = ( s5 + rev(s4) + s6 + rev (s3) + s5 + rev(s1) );

     phrase phrC1 = $C5 <<  A1  << '11111111 11111111 11111111';
     phrase phrG1 = $G5 <<  A1 << '11111111 11111111 11111111';

     phrase phr = phrC1 @@ ( phrC1 ** phrG1);
         print ( phr);
     play (phr);

}


sequence function rev (sequence s ){
   sequence r =  [];
   foreach (int i in s){

      r = [i] + r;
   }
   return r;
}

sequence function news (sequence ns, sequence is ) {
      sequence r = [];
      foreach ( int i in is ) {
                int count = 1;
                foreach ( int n in ns ) {
                  if (count == i ) {
                     r = r + [ n ];
                  }
                  count = count + 1;
                }

      }
   return r;

}
```

B.4 fib.tdf

This program iteratively and recursively calculates the first 10 Fibonnaci numbers and applies it to a note to create some sound.

```
/* Example showing the fibonacci algorithm iteratively and recursively */
```

```
sequence function fib_iter(int n)
{
        int a = 0;
        int b = 1;
        int sum;
        int i;

        sequence s = [0];

        string result = "";


        for(i = 0; i < n; i = i + 1)
        {
                result = result + a;
                s = s + [a];

                /* Don't print the comma on the last one */
                if (i != (n - 1))
                {
                        if (i == n - 2)
                        {
                                result = result + " and ";
                        }
                        else
                        {
                                result = result + ", ";
                        }
                }

                sum = a + b;
                a = b;
                b = sum;
        }

        /* return result */
        return s;
}

int function fib_rec(int n)
{
        if (n <= 1)
        {
                return n;
        }
        else
        {
                return fib_rec(n - 1) + fib_rec(n - 2);
        }
}

int function main()
{
        /*
        print ("The first 20 fibonacci numbers are (using iteration): " +
                fib_iter(20) + "\n");
        print ("The 20th fibonacci number (using recursion) is: " + fib_rec(19) + "\n");
        */

        print ("A musical take on the fibonacci series: ");

        note n = $C4 : 1 // 4;

        print (fib_iter(10));
        print ("\n");
        print ("Starting Note: " ); print (n); print("\n");
        print ("Fibonnaci Applied: ");
        print (n << fib_iter(10));
        print ("\n");
        play (n << fib_iter(10));
}
```

B.5 prime.tdf

This program calculates all of the prime numbers from 0-70, plays them, then reverses the sequence and plays them again. It demonstrates an algorithm for reversing a sequence.

```
/* Example program that finds the prime numbers between 1 and 50 and then
plays them back musically */


/* Multiple return statements */

bool function check_prime(int num){
        if (num < 2)
                return false;
        else
        {
                for (int i = 2; i<num; i = i+1)
                {
                        if ((num % i) == 0)
                                return false;
                }
        }
        return true;
}

sequence function generate_sequence(int num){
        sequence seq = [0];

        for (int i=2; i<num; i = i+1){
                if (check_prime(i))  /* If this is a prime number */
                        seq = seq + [i]; /* Add to sequence */
        }

        return seq;
}
```

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

void function main(){
        sequence s = generate_sequence(70);
        note n1 = $Db3 : 1//16;
        phrase ph1 = n1 << s;
        phrase ph2 = n1 << reverse_sequence(s);
        chord c1 = n1 :: [0, 2, 3, 7, 8, 9];
        play(c1);
        play(ph1 @@ ph2);
}
```

Appendix C

Project Log

This appendix contains the full subversion repository commit log, which shows exactly what was contributed, by whom, and at what time.

```
------------------------------------------------------------------------
r219 | kramkishun | 2010-12-22 18:34:39 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

ran autoindent for ya
------------------------------------------------------------------------
r218 | chatura.atapattu | 2010-12-22 18:29:08 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Do some cleanup styling.  Should need one last auto-indent
------------------------------------------------------------------------
r217 | kramkishun | 2010-12-22 17:55:43 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

autoindented stuff. feel free to revert if you hate it. also feel free to go through
and make stuff look nicer
------------------------------------------------------------------------
r210 | chatura.atapattu | 2010-12-22 12:51:49 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Add commented out line for pulseaudio in execute.ml in case that needs to be used
------------------------------------------------------------------------
r209 | duanemat | 2010-12-22 12:27:46 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/helper.ml
```

```
Removed unused print_scope function from helper.ml.  Every test still works, etc.
------------------------------------------------------------------------
r208 | chatura.atapattu | 2010-12-22 12:18:15 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Update compiler.ml with some more style
------------------------------------------------------------------------
r207 | chatura.atapattu | 2010-12-22 12:04:30 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/printer.ml

Styled printer.ml.  Autoindent required.
------------------------------------------------------------------------
r206 | chatura.atapattu | 2010-12-22 11:59:09 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tonedef.ml

Styled tonedef.ml.  Autoindent required.
------------------------------------------------------------------------
r205 | chatura.atapattu | 2010-12-22 11:51:56 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/helper.ml

Styled helper.ml.  Autoindent required.
------------------------------------------------------------------------
r204 | chatura.atapattu | 2010-12-22 11:32:50 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Reverted compiler.ml to before auto-indent, which broke some shit
------------------------------------------------------------------------
r203 | chatura.atapattu | 2010-12-22 11:25:17 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Fix small bug in execute caused during styling
------------------------------------------------------------------------
r201 | chatura.atapattu | 2010-12-22 11:04:34 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Update style in execute.ml.  Kevin, work your script magic.
------------------------------------------------------------------------
r200 | chatura.atapattu | 2010-12-22 10:22:34 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Update style in compiler.ml.  Kevin, work your script magic.
------------------------------------------------------------------------
r199 | kramkishun | 2010-12-22 10:11:40 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
```

```
ran emacs indent on compiler.. for some reason some stuff still looks a little
funky like when a line is broken up, I think it expects it to be broken up at a
different place or something.
------------------------------------------------------------------------
r198 | chatura.atapattu | 2010-12-22 10:11:23 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli

Style bytecode.mli.  Should be indented, but an auto-indent wouldn't hurt
------------------------------------------------------------------------
r197 | chatura.atapattu | 2010-12-22 09:56:52 -0500 (Wed, 22 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Style compiler.ml without the indenting.  That will be handled automatically.
Kevin, work your emacs magic
------------------------------------------------------------------------
r192 | kramkishun | 2010-12-21 21:24:26 -0500 (Tue, 21 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/scanner.mll

auto indented
------------------------------------------------------------------------
r191 | chatura.atapattu | 2010-12-21 21:19:23 -0500 (Tue, 21 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/scanner.mll

Bring ast, scanner and parser to coding spec
------------------------------------------------------------------------
r190 | kramkishun | 2010-12-21 21:18:47 -0500 (Tue, 21 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/printer.ml

auto indented printer.ml
------------------------------------------------------------------------
r189 | kramkishun | 2010-12-21 21:01:19 -0500 (Tue, 21 Dec 2010) | 1 line
Changed paths:
   M /trunk/docs/Report/report.pdf
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   M /trunk/src/tonedef/scanner.mll
   M /trunk/src/tonedef/tonedef.ml

let emacs autoindent the .ml files with ocaml mode (which uses the standard
ocaml indenting scheme). some things still need to be fixed manually but it
did a lot of good cleaning up and made the doc look nicer too
------------------------------------------------------------------------
r187 | chatura.atapattu | 2010-12-21 19:23:07 -0500 (Tue, 21 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
```

```
Bring ast.mli to the programming style convention
------------------------------------------------------------------------
r176 | duanemat | 2010-12-20 16:15:48 -0500 (Mon, 20 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Reverted back the compiler because it caused variable errors.
------------------------------------------------------------------------
r175 | cah2196 | 2010-12-20 16:10:30 -0500 (Mon, 20 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/samples/demo_intervals.tdf

Added another sample.
------------------------------------------------------------------------
r174 | cah2196 | 2010-12-20 14:38:12 -0500 (Mon, 20 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/samples/brownian.tdf

Changed gold files to match the new string output of jfugue strings instead
of memory representation.
------------------------------------------------------------------------
r173 | chatura.atapattu | 2010-12-20 00:05:21 -0500 (Mon, 20 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/HelloWorld.tdf
   A /trunk/src/tonedef/samples/SmokeOnTheWater.tdf

Update HelloWorld.tdf and add SmokeOnTheWater.tdf.  Smoke on the water is not
complete, the rhythm needs to be applied
------------------------------------------------------------------------
r172 | duanemat | 2010-12-19 23:15:31 -0500 (Sun, 19 Dec 2010) | 2 lines
Changed paths:
   A /trunk/src/tonedef/samples/prime.tdf

Sorry - forgot to commit this earlier.  Just a function that finds the prime
numbers between 0 and 70 and puts them into a sequence, applies to a note and
creates a phrase, then reverses the sequence and creates a new phrase.  The two
phrases are then played consecutively.
Nothing special, but shows off foreach and a couple of other features.
------------------------------------------------------------------------
r171 | duanemat | 2010-12-19 23:12:13 -0500 (Sun, 19 Dec 2010) | 2 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml

A quick fix to the redfinition for functions mentioned earlier.  We can knock it
out, but it was just a few lines of code so I figured it wouldn't hurt.
Passes all other tests.
------------------------------------------------------------------------
r170 | chatura.atapattu | 2010-12-19 23:08:09 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Removed some dead code.  Sorry for all these commits.
------------------------------------------------------------------------
```

```
r169 | chatura.atapattu | 2010-12-19 23:06:31 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Some more minor formatting as I was reading through
------------------------------------------------------------------------
r168 | chatura.atapattu | 2010-12-19 23:03:14 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/scanner.mll

Format the spacing in the code.  I'm a bit OCD and while reading I formatted the
road for more readability.
------------------------------------------------------------------------
r167 | kramkishun | 2010-12-19 18:17:34 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/HelloWorld.tdf
   D /trunk/src/tonedef/samples/Scale.tdf

Updating the sample programs.. HelloWorld didn't compile and Scale.tdf wasn't
valid tonedef.
------------------------------------------------------------------------
r166 | kramkishun | 2010-12-19 15:32:54 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   D /trunk/src/tonedef/dev_playjfugue.ml

removed dev_playjfugue from repo
------------------------------------------------------------------------
r165 | cah2196 | 2010-12-19 15:31:45 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_sub.gold
   M /trunk/src/tonedef/tests/Suite_ops/append_phrases_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/chord_creation_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/note_addition_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_02.gold
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_02.gold
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_03.gold
   M /trunk/src/tonedef/tests/Suite_ops/type_promotion.gold
   M /trunk/src/tonedef/tests/Suite_strings/print_everything.gold

Changed gold files to match the new string output of jfugue strings instead of
memory representation.
------------------------------------------------------------------------
r164 | cah2196 | 2010-12-19 15:31:03 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml
```

```
Changed the promotion of phrases, chords to their jfugue converted strings instead
 of the memory representation.
------------------------------------------------------------------------
r163 | cah2196 | 2010-12-19 10:47:32 -0500 (Sun, 19 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Fixed the function argument type checking. Needed to reverse the list of arg types
to match the order of the evaluation of the arguments.
------------------------------------------------------------------------
r162 | cah2196 | 2010-12-18 17:20:39 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added some comments and cleaned up some sections of the code that aren't needed with
 the type conversion now. Ex: Concat: note, chord case. This is now handled with the
note being converted to a chord in the bytecode before the concat, so the Chord,
Chord case handles it.
------------------------------------------------------------------------
r161 | cah2196 | 2010-12-18 17:05:10 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/samples/quicksort.tdf

Added Sample program quicksort.tdf which sorts a random sequence of integers and
applies the random and sorted sequences to play a phrase.
------------------------------------------------------------------------
r160 | cah2196 | 2010-12-18 17:02:02 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/parser.mly

Added parse match for empty sequences [ ] to parser
------------------------------------------------------------------------
r159 | cah2196 | 2010-12-18 14:45:09 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Updated the phrase to jfugue string conversion to distinguish between spots in the
 phrase that are just padding or are actual rests.
------------------------------------------------------------------------
r158 | cah2196 | 2010-12-18 13:41:15 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Updated the beat to jfugue string conversion.
------------------------------------------------------------------------
r157 | kramkishun | 2010-12-18 13:36:16 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/fib.tdf

fixed samples/fib.tdf to adhere to new rules for sequence addition and added the
play() at the end. it works! woosvn status
```

```
------------------------------------------------------------------------
r156 | cah2196 | 2010-12-18 12:51:26 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Fixed a scoping bug where int x = x; wasn't being caught.
------------------------------------------------------------------------
r155 | duanemat | 2010-12-18 11:35:55 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

Slight change to the beats test.
------------------------------------------------------------------------
r154 | cah2196 | 2010-12-18 11:33:52 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.tdf
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_02.gold
   M /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_03.gold
   M /trunk/src/tonedef/tests/Suite_ops/type_promotion.gold
   M /trunk/src/tonedef/tests/Suite_strings/print_everything.gold

Updated some tests with the updated chord->phrase promotion.
------------------------------------------------------------------------
r153 | cah2196 | 2010-12-18 11:33:10 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Added a bit to chord to phrase conversion in order to pad the phrase slots to make
the entire phrase have the length of the chord. Example: chord [ (N, 1/4)]  -> phrase
[ (N,1/4); [];[];[] ] . The three padding (empty) chords are so that the entire phrase is
1/4 beat long. Each spot in the phrase is a 1/16th duration.
------------------------------------------------------------------------
r152 | cah2196 | 2010-12-18 11:10:16 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_ops/append_phrases_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_02.gold
   A /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_02.tdf
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_02.gold
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_02.tdf
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_03.gold
   A /trunk/src/tonedef/tests/Suite_ops/shift_phrase_test_03.tdf

Added tests for right shift operator.
------------------------------------------------------------------------
r151 | cah2196 | 2010-12-18 11:09:08 -0500 (Sat, 18 Dec 2010) | 1 line
```

```
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added right shift operator for phrases. Fixed a bug in << operator - it wasn't inserting
spaces in the phrase so that each element was a sixteenth duration.
------------------------------------------------------------------------
r150 | duanemat | 2010-12-18 09:29:47 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_sub.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_sub.tdf

Quick subtraction test with gold responses.
------------------------------------------------------------------------
r149 | cah2196 | 2010-12-18 04:47:25 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/scanner.mll

Added rhythms - literals need to be in single quotes to distinguish from regular strings.
2 rhythms can be added. Implemented the phrase<<rhythm operator.
------------------------------------------------------------------------
r148 | cah2196 | 2010-12-18 01:16:40 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added type conversions to binary operators, which allowed me to clean up some of the
execution cases for various bytecodes. For example, adding a beat and int will now convert
the int to a beat and then use the beat+beat computation in execute. And similarly for some other ops.
------------------------------------------------------------------------
r147 | chatura.atapattu | 2010-12-18 01:15:36 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml
   M /trunk/src/tonedef/execute.ml

Integrate play_jfugue and phrase conversion into execute. A phrase on the top of the stack
called with play should play. Will look into a test case tomorrow. Code compiles and all existings tests
------------------------------------------------------------------------
r146 | chatura.atapattu | 2010-12-18 00:22:54 -0500 (Sat, 18 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

Update mbeat_to_jfugue to handle pretty much any combination of numerator and denominator
------------------------------------------------------------------------
r145 | duanemat | 2010-12-17 23:06:10 -0500 (Fri, 17 Dec 2010) | 5 lines
```

```
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/tests/Suite_logic/comparison_ops_types.gold
   M /trunk/src/tonedef/tests/Suite_logic/comparison_ops_types.tdf

Added support for comparison of ints, booleans, beats, pitches, and notes.  Also updated tests.

Right now, I just return a raw 1/0 value for two Notes regardless of the actual comparison
operator run on it - in other words, n1 < n2 gives you the same result as n1 = n2.  I will
need to go through and update the potential operators for each data type.

Right now that means ints and beats allow < > >= ..., while pitches, booleans, and notes are
limited to == and !=.  That makes sense, but just need to implement it.
------------------------------------------------------------------------
r144 | cah2196 | 2010-12-17 22:33:28 -0500 (Fri, 17 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Added checks that the predicate expression to a while or if is convertable to Boolean. Throws
failure if program has something like if(pitch_var) {...}
------------------------------------------------------------------------
r143 | chatura.atapattu | 2010-12-17 22:12:15 -0500 (Fri, 17 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

Update dev_playjfugue to handle null chords in a phrase
------------------------------------------------------------------------
r142 | duanemat | 2010-12-17 22:07:08 -0500 (Fri, 17 Dec 2010) | 8 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   A /trunk/src/tonedef/tests/Suite_logic/comparison_ops_types.gold
   A /trunk/src/tonedef/tests/Suite_logic/comparison_ops_types.tdf

Couple of changes:
Added error for beats with x//0
Have some of the comparison operators working:
- Int, Boolean, Beat, Pitch.

Still a work in progress, but I figured some functionality would be nice.

Also added tests.
------------------------------------------------------------------------
r141 | cah2196 | 2010-12-17 22:06:13 -0500 (Fri, 17 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/scanner.mll

Changed regex for string literals to match \" in the middle of a string.
------------------------------------------------------------------------
r140 | duanemat | 2010-12-17 11:32:53 -0500 (Fri, 17 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_strings/print_everything.gold
   A /trunk/src/tonedef/tests/Suite_strings/print_everything.tdf

Quick test that prints out every possible data type we have in the LRM (save rhythms,
which we may drop).  Will hopefully catch the situation we ran into earlier with all
the promotion changes.
------------------------------------------------------------------------
r139 | duanemat | 2010-12-16 14:46:33 -0500 (Thu, 16 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/helper.ml

Allow sequence to be promoted to string.  Passed all tests + fib and Hello World.
------------------------------------------------------------------------
r138 | duanemat | 2010-12-16 10:35:21 -0500 (Thu, 16 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Figured out problem - not sure why though.  Apparently when you raise/lower pitch
octaves, the Note->Pitch conversion was fired off because the promote function in helper
allows such a promotion implicitly.  This created a situation where note->pitch actually
returned a note of just the pitch, which was a corner case that I didn't check for.
It now works, and passes all of our tests.
------------------------------------------------------------------------
r137 | duanemat | 2010-12-16 01:23:06 -0500 (Thu, 16 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   A /trunk/src/tonedef/tests/Suite_ops/type_promotion.gold
   A /trunk/src/tonedef/tests/Suite_ops/type_promotion.tdf

Implemented ConverTtype function in execute.ml.  Inexplicably breaks the raise_octave
tests for reasons that are not readily apparent to me at 1:30 am, but I will look at
them more closely tomorrow morning.  My own tests all work, though again I'll have to
check tomorrow to see if there are any glaring mistakes.  Also going to try to
streamline code a bit.
------------------------------------------------------------------------
r136 | chatura.atapattu | 2010-12-15 16:56:40 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Update makefile to remove dev_playjfugue. dev_playjfugue is just for development and
will break tonedef and the tests until properly integrated
------------------------------------------------------------------------
r135 | chatura.atapattu | 2010-12-15 16:52:28 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

Minor changes to dev_playjfugue
------------------------------------------------------------------------
r134 | kramkishun | 2010-12-15 16:41:10 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

Works as separate functions now
```

```
------------------------------------------------------------------------
r133 | duanemat | 2010-12-15 16:40:04 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Updated Makefile so that it compiles with dev_playjfugue.
------------------------------------------------------------------------
r132 | kramkishun | 2010-12-15 16:34:53 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

got dev_playjfugue to compile
------------------------------------------------------------------------
r131 | chatura.atapattu | 2010-12-15 16:28:08 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/dev_playjfugue.ml

Update dev_playjfugue with code that is functional in the interpreter.  Working on
making a fully compiling file
------------------------------------------------------------------------
r130 | cah2196 | 2010-12-15 16:22:47 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Let everything be promotable to strings so that they can be printed with the new type
checking.
------------------------------------------------------------------------
r129 | cah2196 | 2010-12-15 15:44:47 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Added Type conversion checking to compiler on assignment, fncall, returns, sequences
------------------------------------------------------------------------
r128 | cah2196 | 2010-12-15 14:53:39 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Fixed scope checking issue - wasn't recursing on the expressions in the FnCall
argument list to make sure that x is in scope for something like "print (x);"
------------------------------------------------------------------------
r127 | cah2196 | 2010-12-15 14:11:27 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Added type checking/conversion before the Store opcode in assignment
------------------------------------------------------------------------
r126 | cah2196 | 2010-12-15 13:31:36 -0500 (Wed, 15 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/helper.ml

Fixed the bug that void functions couldn't use return to terminate.
------------------------------------------------------------------------
r125 | kramkishun | 2010-12-14 21:12:05 -0500 (Tue, 14 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/fib.tdf

Applied the fibonacci numbers (first 10) to a note to build up a phrase. Let me
know what you guys think.
------------------------------------------------------------------------
r124 | kramkishun | 2010-12-14 21:05:47 -0500 (Tue, 14 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/fib.tdf

Updated the fib.tdf sample to use sequences
------------------------------------------------------------------------
r123 | kramkishun | 2010-12-14 20:57:09 -0500 (Tue, 14 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Implemented + for ints and sequences, if you do 1 + [2,3,4,5] it'll return [1,2,3,4,5].
if you do [2,3] + 4, you get [2,3,4], etc. We can use this to build up sequences
using an algorithm (say fibonacci?) and then apply it to a note to make
 a phrase using the <<
------------------------------------------------------------------------
r122 | duanemat | 2010-12-13 23:46:09 -0500 (Mon, 13 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

beat/int mixing - completed binops and updated tests.
------------------------------------------------------------------------
r121 | duanemat | 2010-12-13 22:33:10 -0500 (Mon, 13 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

Added add/sub code for mixed beats + ints and tests.  Will implement
multiplication and division as well.
------------------------------------------------------------------------
r120 | kramkishun | 2010-12-13 22:29:48 -0500 (Mon, 13 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

Slight tweak to test.py to make output easier to read and keep count of tests
------------------------------------------------------------------------
r119 | kramkishun | 2010-12-13 22:20:17 -0500 (Mon, 13 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_ops/append_phrases_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/append_phrases_test_01.tdf

Implemented @@, which can append two phrases. Also added test for @@.
------------------------------------------------------------------------
r118 | kramkishun | 2010-12-13 21:40:11 -0500 (Mon, 13 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/phrase_creation_test_01.tdf

Implemented and added tests for the << operator, which can now create phrases
from notes and sequences. The other version of << which operates on rhythms has
NOT been implemented. Also added the ability to print a phrase just by doing print(p),
but concatenate does not work on phrases + strings yet. We CAN, however, do note <<
[1,2,3,4] or something like that, and a phrase will be created. This is probably enough
to start working on a function that can play phrases. See the test file for more details.
------------------------------------------------------------------------
r117 | duanemat | 2010-12-12 22:05:08 -0500 (Sun, 12 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_mod.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_mod.tdf

Added modulo operator (%) for ints + tests.  Had to add Modx to bytecode
because we already have Mod in AST.
------------------------------------------------------------------------
r116 | duanemat | 2010-12-12 20:52:36 -0500 (Sun, 12 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

Beat - implemented multiplication, division, and tests.  Had to update parser
because beat creation had wrong precedence, creating issue with multi/div implementation.
------------------------------------------------------------------------
r115 | chatura.atapattu | 2010-12-12 10:40:15 -0500 (Sun, 12 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/dev_playjfugue.ml

Add file for the development of a function to parse a phrase into a jfugue string.
Will probably eventually be incorporated into helper or similar file
------------------------------------------------------------------------
r114 | kramkishun | 2010-12-11 18:56:13 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
```

```
Got rid of unused match case that was causing a warning.
------------------------------------------------------------------------
r113 | kramkishun | 2010-12-11 15:44:00 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/tests/Suite_ops/note_addition_test_01.gold
   M /trunk/src/tonedef/tests/Suite_ops/note_addition_test_01.tdf

Added + for chords; test case
------------------------------------------------------------------------
r112 | kramkishun | 2010-12-11 12:19:02 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/samples/fib.tdf

Added a 'sample' program that computes the fibonacci sequence both iteratively
and recursively. This might be something we can show Edwards because our language
officially does more than microc lol. It demonstrates if/else, for looping, function
calling and returning parameters, and recursion. Runs perfectly with ./tonedef
< samples/fib.tdf
------------------------------------------------------------------------
r111 | duanemat | 2010-12-11 12:10:29 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

Added subtractions of beats.
------------------------------------------------------------------------
r110 | kramkishun | 2010-12-11 11:52:19 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Added concatenation of integers and strings
------------------------------------------------------------------------
r109 | kramkishun | 2010-12-11 11:38:16 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Commented out the circular reference to str.cma in the makefile, not sure what
that was for but make kept giving a warning.
------------------------------------------------------------------------
r108 | kramkishun | 2010-12-11 11:31:01 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli

Changed the comment in bytecode.mli to reflect how I used CreateChrd bytecode.
I used it for both creating a chord from 2 notes or creating a chord from a note
and a chord. I just realized that maybe 'Combine' would've been a better bytecode
 to use so that can be switched over very easily if you guys want.
------------------------------------------------------------------------
r107 | kramkishun | 2010-12-11 11:22:18 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_ops/chord_creation_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/chord_creation_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_ops/note_addition_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/note_addition_test_01.tdf

Fixed the order in which chords were created from notes (doesn't matter for chords
but will matter for phrases) and added two tests: one for + on notes and chords and
 one for ::
------------------------------------------------------------------------
r106 | kramkishun | 2010-12-11 10:48:32 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Got :: working properly, its amazing what 8 hours of sleep can accomplish lol
------------------------------------------------------------------------
r105 | kramkishun | 2010-12-11 02:16:59 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Began implementation of ::, I can create a chord from the notes duplicated the
same number of times as there are elements in the sequence, but for some reason I
 can't apply the sequence to raise the pitch of each note. Will check it out tomorrow
when I get some sleep. For now the changes are checked in because all the tests
still run fine and maybe you guys can take a crack at it.
------------------------------------------------------------------------
r104 | duanemat | 2010-12-11 00:43:28 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_beat.tdf

Re-implemented negation of beats after Kevin's change.  Added beat tests and gold.
------------------------------------------------------------------------
r103 | kramkishun | 2010-12-11 00:31:45 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Added support for creating chords from more than 2 notes (ex. n1 + n2 + n3 + n4
creates a chord with 4 notes). Can also do things like add a chord and a note to get a new chord.
------------------------------------------------------------------------
r102 | kramkishun | 2010-12-11 00:21:22 -0500 (Sat, 11 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Fixed the storage of beats on the stack and added support for the + operation on
 two notes, which creates a chord. Also added printing of chords (ex. note n1 =  :
 3//4; note n2 =  : 2 //3; print (n1 + n2); works correctly)
------------------------------------------------------------------------
r101 | kramkishun | 2010-12-10 22:55:53 -0500 (Fri, 10 Dec 2010) | 1 line
```

```
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_andor_01.gold
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_andor_01.tdf

implemented boolean AND and OR. added test case
------------------------------------------------------------------------
r100 | cah2196 | 2010-12-10 21:51:58 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_ops
   A /trunk/src/tonedef/tests/Suite_ops/raise_note_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/raise_note_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_ops/raise_octave_test_01.gold
   A /trunk/src/tonedef/tests/Suite_ops/raise_octave_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_ops/raise_octave_test_02.gold
   A /trunk/src/tonedef/tests/Suite_ops/raise_octave_test_02.tdf

Added tests for RaiseNote, RaiseOctave
------------------------------------------------------------------------
r99 | duanemat | 2010-12-10 21:22:21 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Fixed problem with how beats were stored on stack.  Compiled and passes all current tests.
------------------------------------------------------------------------
r98 | cah2196 | 2010-12-10 20:56:43 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added pitch translation, RaiseNote & RaiseOctave operators. Found and fixed a bug
with unops - they were storing the result into wrong spot on stack.
------------------------------------------------------------------------
r97 | cah2196 | 2010-12-10 17:50:12 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.gold
   M /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.tdf
   A /trunk/src/tonedef/tests/Suite_loops
   A /trunk/src/tonedef/tests/Suite_loops/loops_test_01.gold
   A /trunk/src/tonedef/tests/Suite_loops/loops_test_01.tdf

Added test suite for loops, and a test for foreach loop on sequence.
------------------------------------------------------------------------
r96 | cah2196 | 2010-12-10 17:48:46 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added translation/execution of defining sequences and doing foreach loops on sequences.
```

```
------------------------------------------------------------------------
r95 | cah2196 | 2010-12-10 00:09:12 -0500 (Fri, 10 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml

Added Create* and Combine opcodes to execute.ml. Added translation of BuildNote(:)
operator to compiler.ml.
------------------------------------------------------------------------
r94 | duanemat | 2010-12-09 09:45:46 -0500 (Thu, 09 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli

Sorry - failed to include the bytecode.mli file.
------------------------------------------------------------------------
r93 | duanemat | 2010-12-09 09:44:06 -0500 (Thu, 09 Dec 2010) | 2 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Beats:  Implemented PushBeat that pushes a tuple of a beat onto the stack.  Also am
working with addition and subtraction.
For testing, I create a function called show_stack that prints out the current
contents of the stack.
------------------------------------------------------------------------
r92 | kramkishun | 2010-12-07 22:24:09 -0500 (Tue, 07 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/LRM_test.syntax (from /trunk/src/tonedef/tests/LRM_test.tdf:91)
   D /trunk/src/tonedef/tests/LRM_test.tdf

Just renamed the LRM_test.tdf to a .syntax because it was interfereing with
the test script (redeclared variables and stuff, anyway that test is outdated and isn't executable)
------------------------------------------------------------------------
r91 | duanemat | 2010-12-07 21:55:20 -0500 (Tue, 07 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_02.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_02.tdf
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_03.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_03.tdf

Compiler/Execute:  Added multiplication and division of ints + tests.
------------------------------------------------------------------------
r90 | duanemat | 2010-12-07 15:09:34 -0500 (Tue, 07 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_unop_01.tdf

Added negation of ints and inverse of bool plus test suite.
```

```
------------------------------------------------------------------------
r89 | kramkishun | 2010-12-06 22:19:57 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

test.py runs make if its not compiled yet
------------------------------------------------------------------------
r88 | chatura.atapattu | 2010-12-06 20:27:43 -0500 (Mon, 06 Dec 2010) | 2 lines
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.gold
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.gold
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.tdf
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_for01.gold
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_01.tdf
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_02.tdf
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_03.tdf
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_04.tdf

Fix newlines in subfolders of test.
Remove extra line in Makefile.
------------------------------------------------------------------------
r87 | chatura.atapattu | 2010-12-06 20:21:33 -0500 (Mon, 06 Dec 2010) | 3 lines
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/tests/LRM_test.tdf
   M /trunk/src/tonedef/tests/LRM_test.txt
   M /trunk/src/tonedef/tests/hello_world.tdf
   M /trunk/src/tonedef/tests/simplest.txt

Update makefile to have a "test" command.  Run "make test" to compile and run
the test script.  "make" will only build the tonedef executable.

Stripped the files in test folder of "\r" so that all newlines are simply "\n"
------------------------------------------------------------------------
r86 | kramkishun | 2010-12-06 20:11:35 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.gold
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_03.tdf

Some of the tests' expected outputs were incorrect
------------------------------------------------------------------------
r85 | kramkishun | 2010-12-06 20:07:39 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

Some of the tests were failing because the .gold files had newlines ending with
\r\n and tonedef was outputting newlines as \n, so I added code to strip out the
\r in all of the files so it'll work on both windows and *nix. I also added a -v
switch to the script, so if you do 'python test.py -v' it will show you the
expected output and actual output of the files that failed
------------------------------------------------------------------------
r84 | kramkishun | 2010-12-06 19:51:10 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/hello_world.gold (from /trunk/src/tonedef/tests/hello_world.out:83)
   D /trunk/src/tonedef/tests/hello_world.out
   A /trunk/src/tonedef/tests/simple_var.gold (from /trunk/src/tonedef/tests/simple_var.out:83)
   D /trunk/src/tonedef/tests/simple_var.out

Renamed the .out files to .gold so the test script would pick them up
------------------------------------------------------------------------
r83 | kramkishun | 2010-12-06 19:45:28 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/test.py

Added the auto test script to the Makefile. It'll run whenever we run 'make'
right after everything is compiled so we can see if we broke anything.
You guys can remove it if you don't want it there.
------------------------------------------------------------------------
r82 | kramkishun | 2010-12-06 19:42:02 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

Small update to test.py to make the output consistent.
------------------------------------------------------------------------
r81 | kramkishun | 2010-12-06 19:38:58 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

test.py officially works. Run it with 'python test.py'. It will traverse the
tests directory and report which files pass, fail, or error. A pass means that
 tonedef ran the code and the output matched the .gold file. A fail means that
tonedef ran the code and the output did not match the .gold file, and an error
means that tonedef could not run the code. Make sure you run 'make' first or it'll complain
------------------------------------------------------------------------
r80 | duanemat | 2010-12-06 15:36:03 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.gold
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.tdf

Minor update:  Updated logic test to show functionality for if...(no else)/
dangling else statement.
------------------------------------------------------------------------
r79 | duanemat | 2010-12-06 15:34:12 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_for01.gold
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_for01.tdf

Compiler: for loop - for loop implemented + test suite.
------------------------------------------------------------------------
r77 | kramkishun | 2010-12-06 00:31:48 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

If a .gold file does not exist for a .tdf, it now only issues a warning and skips
that test. Only errors if tonedef output does not match or if tonedef does not run
```

```
properly.
------------------------------------------------------------------------
r76 | kramkishun | 2010-12-06 00:29:13 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

Added some more error checking to test.py for it to make sure that a gold file
 exists for the tdf file its testing
------------------------------------------------------------------------
r75 | kramkishun | 2010-12-06 00:06:56 -0500 (Mon, 06 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

fiddling around with test.py to get it to run properly. we need to make sure for
every .tdf file that exists in the tests/ directory, there exists a .gold file with the same name
------------------------------------------------------------------------
r74 | duanemat | 2010-12-05 23:53:54 -0500 (Sun, 05 Dec 2010) | 6 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.gold
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_01.tdf
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.gold
   M /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.tdf

Update:  ifthenelse and while loops now working.
1.  Updated compiler.ml with Bytecode.Load and Bytecode.PushBool (they were just Load and (Pushbool)
2.  Updated execute with the correct offsets
3.  added MemBool match to int_value_of_mem.
4.  Added some logic tests + .gold.

------------------------------------------------------------------------
r73 | duanemat | 2010-12-05 21:15:02 -0500 (Sun, 05 Dec 2010) | 3 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Compiler ifthenelse - Generates proper bytecode for if...else... - execute.ml
is still not working properly.  Seems to be a memory issue.  Will figure it out.

Also added line counter to print_bp in helper for debugging.  Can obviously remove
 later, but it was helpful for me to figure out why loop was skipping.
------------------------------------------------------------------------
r72 | duanemat | 2010-12-05 19:42:15 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.gold
   A /trunk/src/tonedef/tests/Suite_logic/suite_logic_test_02.tdf

Test case for while loop.
------------------------------------------------------------------------
r71 | duanemat | 2010-12-05 17:16:48 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
```

```
Implemented compiler:while loop - While loop is somewhat implemented.
Logic seems screwy, though, so I'm working on it.  Figured I would update
for now so that others have it.
------------------------------------------------------------------------
r70 | cah2196 | 2010-12-05 17:07:09 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   M /trunk/src/tonedef/scanner.mll

Implemented translation/execution of comparison functions on Ints. Can now say
"bool x = 5 < 4; print (x)" and it prints "false". I collapsed the individual
bytecodes for Gt,Lt,... into one that is Compare (bop) and the matching is done
on bop in Execute.ml. There is also a MemBool(b) tag in the memory union.
------------------------------------------------------------------------
r69 | cah2196 | 2010-12-05 17:01:07 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_logic/comparison_test_01.gold
   A /trunk/src/tonedef/tests/Suite_logic/comparison_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_logic/logic_test_01.gold
   A /trunk/src/tonedef/tests/Suite_logic/logic_test_01.tdf

Added tests for testing the compare ops on ints
------------------------------------------------------------------------
r68 | cah2196 | 2010-12-05 15:50:55 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_01.gold
   A /trunk/src/tonedef/tests/Suite_arithmetic/arithmetic_test_01.tdf

Added integer addition translation/execution.
------------------------------------------------------------------------
r67 | cah2196 | 2010-12-05 15:50:35 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml
   M /trunk/src/tonedef/printer.ml

Added integer addition translation/execution.
------------------------------------------------------------------------
r66 | cah2196 | 2010-12-05 11:45:30 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/scanner.mll

Added boolean literals true and false to AST, scanner, parser.
------------------------------------------------------------------------
r65 | cah2196 | 2010-12-05 01:58:30 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_03.gold
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_03.tdf
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_04.gold
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_04.tdf

Added string concatenation translation/execution, and more tests. Also, commented
out the automatic playing of notes in execute.ml. But it works, which is awesome!
------------------------------------------------------------------------
r64 | cah2196 | 2010-12-05 01:58:02 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Added string concatenation translation/execution, and more tests. Also, commented o
ut the automatic playing of notes in execute.ml. But it works, which is awesome!
------------------------------------------------------------------------
r63 | cah2196 | 2010-12-05 01:57:38 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Added string concatenation translation/execution, and more tests. Also, commented
out the automatic playing of notes in execute.ml. But it works, which is awesome!
------------------------------------------------------------------------
r62 | kramkishun | 2010-12-05 01:04:44 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/test.py

slight change to the test script now that tonedef.ml works. still needs a little bit of work
------------------------------------------------------------------------
r61 | cah2196 | 2010-12-05 00:59:36 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/execute.ml

Changed print_endline to print_string in Jmp(-1) case
------------------------------------------------------------------------
r60 | kramkishun | 2010-12-05 00:53:01 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tonedef.ml

Updated tonedef.ml to add some switches. Now tonedef -b prints the bytecode of
a program, tonedef -e executes the program and tonedef -p prints the tokens. By
default, tonedef alone will execute the program. An example usage to print
bytecode: './tonedef -b < file.tdf' or to just execute './tonedef < file.tdf'
------------------------------------------------------------------------
r59 | kramkishun | 2010-12-05 00:18:53 -0500 (Sun, 05 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/tonedef.ml

Fixed tonedef.ml to compile. Had to also make an adjustment to the Makefile so that
 both PlayJFugue and tonedef can be compiled in one 'make' command.
Tonedef plays a scale in JFugue and outputs bytecode
```

```
------------------------------------------------------------------------
r58 | kramkishun | 2010-12-04 23:52:24 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/test.py

Added a preliminary test script that'll traverse through the tests, run the .tdf
 file and compare output against the .gold files. Didn't get to test yet
because tonedef.ml won't compile.
------------------------------------------------------------------------
r57 | kramkishun | 2010-12-04 22:25:29 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   D /trunk/src/tonedef/bytecode.ml

Deleted bytecode.ml because bytecode.mli is more up to date and is the one being used
------------------------------------------------------------------------
r56 | chatura.atapattu | 2010-12-04 21:57:42 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/PlayJFugue.java
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/java/PlayJFugue.java

Update PlayJFugue.java in main dir and java dir.  Add function call to execute.ml,
which causes a static music string.  Update makefile to have required unix flags
------------------------------------------------------------------------
r55 | cah2196 | 2010-12-04 21:22:22 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_01.tdf
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_02.tdf

Minor changes to tests - added new line and removed a double variable declaration.
------------------------------------------------------------------------
r53 | chatura.atapattu | 2010-12-04 19:19:51 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/PlayJFugue.java
   M /trunk/src/tonedef/java/PlayJFugue.java
   A /trunk/src/tonedef/jfugue.jar

Update PlayFugue.java in tonedef/java.  Copy PlayJFugue.java to main tonedef foler.
Update makefile to make PlayJFugue.
------------------------------------------------------------------------
r52 | cah2196 | 2010-12-04 19:14:53 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/scanner.mll

Added process of strings before they become a  string literal token in the scanner.
This converts the escape character sequences and removes the surrounding quotes.
------------------------------------------------------------------------
r51 | chatura.atapattu | 2010-12-04 18:34:36 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/java/Makefile

Make makefile compatible for linux and OSX, checking which OS is running and
```

```
setting to use PulseAudio or not.  In linux, pulseaudio is required.
In OSX, should work out of the box
------------------------------------------------------------------------
r50 | chatura.atapattu | 2010-12-04 17:08:13 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/java/MIDITest.java

Add MIDITest java program
------------------------------------------------------------------------
r49 | cah2196 | 2010-12-04 16:52:52 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Made a changes to internal function translate_expr. There is now a type typed_code
that has members typ that is a td_type and code that is a bytecode list. There is now
a translate_type_expr that creates a typed_code translation for the expression. The typ i
n the typed_code struct is the type of the value that will be on top of the stack after
the corresponding code is executed. This will let us know types of subexpressions that we are translatin
------------------------------------------------------------------------
r48 | cah2196 | 2010-12-04 15:17:00 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml

Changed Return(expr) to Return (expr option) in AST. Added a line to parser.mly
to parse RETURN SEMICOLON to Return(None). Modified the Return(x) matches
in compiler.ml, helper.ml, printer.ml to comply with new type.
------------------------------------------------------------------------
r47 | cah2196 | 2010-12-04 13:56:07 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/execute.ml
   M /trunk/src/tonedef/helper.ml

Added to Execute.exec for Enty and Return cases
------------------------------------------------------------------------
r46 | cah2196 | 2010-12-04 12:26:27 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Added FnCall case to Compiler.ml: translate_expr
------------------------------------------------------------------------
r45 | cah2196 | 2010-12-04 11:48:33 -0500 (Sat, 04 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/helper.ml

Added Helper.ml:string_of_bc which converts bytecode to string, Helper.ml:print_bp
which prints a bytecode array, and Compiler.ml:write_bp_text which outputs a bytecode array to file as t
------------------------------------------------------------------------
r44 | duanemat | 2010-12-03 22:45:18 -0500 (Fri, 03 Dec 2010) | 1 line
```

```
Changed paths:
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_01.tdf
   M /trunk/src/tonedef/tests/Suite_strings/strings_test_02.tdf

Fixed the comments so that they are not (* but /*
------------------------------------------------------------------------
r43 | chatura.atapattu | 2010-12-03 16:28:21 -0500 (Fri, 03 Dec 2010) | 2 lines
Changed paths:
   M /trunk/src/tonedef/java/Makefile
   A /trunk/src/tonedef/java/PlayJFugue.java

Update makefile to work only in windows (apparently to get it working in linux,
there are required MIDI hacks).
Add PlayJFugue program that will be used to call JFugue from within the tonedef
OCaml interpreter.
------------------------------------------------------------------------
r42 | duanemat | 2010-12-03 13:52:08 -0500 (Fri, 03 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Sorry - forgot to include execute.cmo to the Makefile.  Still doesn't fix my earlier
issue, but for completeness.
------------------------------------------------------------------------
r41 | duanemat | 2010-12-03 13:48:28 -0500 (Fri, 03 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Updated Makefile that deals with dependency issues for compiler.  The new code that
Curtis submitted broke my version of tonedef.ml (which should be the current version).
 It looks like our definition for translate in compiler.ml is the issue - I will take a look at it.
------------------------------------------------------------------------
r40 | cah2196 | 2010-12-03 13:23:18 -0500 (Fri, 03 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests/Suite_arithmetic
   A /trunk/src/tonedef/tests/Suite_logic
   A /trunk/src/tonedef/tests/Suite_strings
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_01.gold
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_01.tdf
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_02.gold
   A /trunk/src/tonedef/tests/Suite_strings/strings_test_02.tdf

Adding folders in test for different test suites, just to keep things organized
and less cluttered as we add more tests. There are 2 tests in the Suite_strings
so far. I've created <testname>.gold files for the expected output text to compare
results of our system to. Also, I've put comments with an extra asterisk
 (**  purpose of the test **) in the test files figuring we might want incorporate
that info into a log when running many tests from a script. If anyone has any suggestions
for/problems with these guidelines, let's discuss it before we start writing/commiting too many.
------------------------------------------------------------------------
r39 | cah2196 | 2010-12-03 13:08:41 -0500 (Fri, 03 Dec 2010) | 5 lines
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Separated write_bp and translate into 2 functions. So now can do (with prog: Ast.program):
```

```
let bprog = Compiler.translate prog
in
Compiler.write_bp filename bprog;
Execute.run bprog
------------------------------------------------------------------------
r38 | cah2196 | 2010-12-03 12:43:00 -0500 (Fri, 03 Dec 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/execute.ml

Adding a bare-bones execute.ml. Only processes a few bytecodes as of now.
------------------------------------------------------------------------
r37 | cah2196 | 2010-12-03 08:19:54 -0500 (Fri, 03 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/bytecode.mli
   M /trunk/src/tonedef/compiler.ml
   A /trunk/src/tonedef/helper.ml
   A /trunk/src/tonedef/interpreter.ml
   M /trunk/src/tonedef/tonedef.ml

Made a copy of compiler.ml as interpreter.ml because it was built off of Edwards'
interpreter. Made compiler.ml file fit more to his compiler.ml for microc. Made a
 helper.ml that has function definitions for use in compiler in an attempt to add
modularity to the code and make it less confusing. In helper, there is function that
steps through the AST and checks for scoping issues and type issues although it is not
 complete, it compiles. There are more checks to implement. Also, moved bytecode.ml
to an mli because it is just type definitions like ast.mli. Everything compiles up to
 tonedef.exe when you make.
------------------------------------------------------------------------
r36 | duanemat | 2010-12-02 22:23:55 -0500 (Thu, 02 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/compiler.ml
   A /trunk/src/tonedef/tests/LRM_test.tdf
   A /trunk/src/tonedef/tests/hello_world.out
   A /trunk/src/tonedef/tests/hello_world.tdf
   A /trunk/src/tonedef/tests/simple_var.out
   A /trunk/src/tonedef/tests/simple_var.tdf

We now have a quasi-working print statement working along with some variable storage.
 I have definitely not tested it to any major extent, but I did run a couple of very simple
tests and they produced what I thought should be the output (I did wind up using a
reg exp for removing the quotes around strings - this seems like we might need a change
 down the line).  No binop stuff yet, but that is next on the list.
------------------------------------------------------------------------
r35 | cah2196 | 2010-12-02 22:11:35 -0500 (Thu, 02 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/bytecode.ml

Modified/Added to the bytecode definition. Added types for represententing our data
 as a stack of memory
------------------------------------------------------------------------
r34 | kramkishun | 2010-12-02 20:19:10 -0500 (Thu, 02 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/bytecode.ml

Added a bytecode file that contains the master type definition for all of the bytecode
instructions that we will be generating.
------------------------------------------------------------------------
r33 | duanemat | 2010-12-01 12:17:04 -0500 (Wed, 01 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/compiler.ml

Updated compiler based on changes made by Curtis.  It will compile correctly and runs.
We should probably sit down after class today and figure out how to move forward.
------------------------------------------------------------------------
r32 | cah2196 | 2010-12-01 11:37:58 -0500 (Wed, 01 Dec 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml

Changing these back to previous version. We need function bodies to be just a stmt because
{ ... } parses to a stmt, namely Block(stmts). If we let the bodies be stmts, then we aren't
 forcing function bodies to be in brackets. Did not update compiler.ml to work
with the reverted AST because it is new to me and not sure what needs to change.
------------------------------------------------------------------------
r31 | duanemat | 2010-11-30 22:58:36 -0500 (Tue, 30 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/tests/LRM_test.txt
   A /trunk/src/tonedef/tests/simplest.txt

Here is a slowly-evolving compiler that will parse and store variables and
 perform some rudimentary name checking.  Clearly not much in the way of
functionality, but will have more up tomorrow.  Noticed that our function
definitions in the AST were limited to stmt, not stmts.  Seems to make more
 sense as a list of statements, so made those changes to the various elements.
We can discuss if anyone thinks that isn't necessary.  Thanks.
------------------------------------------------------------------------
r30 | duanemat | 2010-11-30 13:43:50 -0500 (Tue, 30 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/compiler.ml
   M /trunk/src/tonedef/tonedef.ml

This is a bare-bones compiler.ml and an updated tonedef.ml that allows you either
 print out the code or run the compiler.  The compiler did more when I was trying to use
lists, but I'm just reworking it right now.  I will update later today, but I
figured I would put it in the repository so that we have it available.
Go crazy if you guys have any ideas.
------------------------------------------------------------------------
r29 | cah2196 | 2010-11-28 21:39:40 -0500 (Sun, 28 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/parser.mly

Updated parser.mly, scanner.mll, ast.mli, printer.ml. Resolved all conflicts with
the grammar. Reordered the precedence rules to match LRM. Added dangling else
disambiguation. Ran the test/LRM_test.txt through tonedef.exe and it cleanly parsed and printed.
------------------------------------------------------------------------
r28 | cah2196 | 2010-11-28 15:18:53 -0500 (Sun, 28 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/scanner.mll
   M /trunk/src/tonedef/tests/LRM_test.output

Updated parser.mly, scanner.mll, ast.mli, printer.ml. Resolved all conflicts with
the grammar. Reordered the precedence rules to match LRM. Added dangling else disambiguation.
 Ran the test/LRM_test.txt through tonedef.exe and it cleanly parsed and printed.
------------------------------------------------------------------------
r27 | duanemat | 2010-11-02 22:32:38 -0400 (Tue, 02 Nov 2010) | 18 lines
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/scanner.mll
   M /trunk/src/tonedef/tests/LRM_test.output
   M /trunk/src/tonedef/tests/LRM_test.txt

So these are the changes that I have made, plus some issues that we still need to address:

Changes Made:

1.  Added _ character as an option to begin identifier
2.  Added [] to scanner
3.  Added escape sequences
4.  Added "Void" to parser and also created special case for "main" function
5.  Fixed parsing for pitchlit.  Was not working with # and _
6.  Added % functionality.
7.  Added boolean logic parsing for && and ||


Remaining issues:
1.  Parse error with phrases, rhythms, etc. with strings in the expression.
2.  Nested functions do not work yet.
3.  Seems to die with { } standing alone.
4.  Parse errors for if-else
------------------------------------------------------------------------
r26 | duanemat | 2010-11-02 15:52:12 -0400 (Tue, 02 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   M /trunk/src/tonedef/scanner.mll
   A /trunk/src/tonedef/tests/LRM_test.output
   A /trunk/src/tonedef/tests/LRM_test.txt
```

```
Made a couple of changes as I checked our parser/scanner/ast vs. the LRM.  Still
working on more changes, but figured I would upload what I have.  Also uploaded my
 LRM_test.txt file that includes everything I've tested successfully, along with proper output.
------------------------------------------------------------------------
r25 | chatura.atapattu | 2010-11-01 11:53:21 -0400 (Mon, 01 Nov 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/samples/HelloWorld.tdf
   M /trunk/src/tonedef/samples/Scale.tdf

Updated sample programs to reflect "main" function definition based on LRM
------------------------------------------------------------------------
r24 | chatura.atapattu | 2010-10-31 17:10:47 -0400 (Sun, 31 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/java/Scale.java
   A /trunk/src/tonedef/samples
   A /trunk/src/tonedef/samples/HelloWorld.tdf
   A /trunk/src/tonedef/samples/Scale.tdf

Add some sample programs
------------------------------------------------------------------------
r23 | chatura.atapattu | 2010-10-31 16:23:13 -0400 (Sun, 31 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/java/Makefile

Fix makefile for java programs
------------------------------------------------------------------------
r22 | chatura.atapattu | 2010-10-31 16:19:17 -0400 (Sun, 31 Oct 2010) | 1 line
Changed paths:
   A /trunk/docs/jfugue-chapter1.pdf (from /trunk/src/tonedef/java/jfugue-chapter1.pdf:21)
   A /trunk/docs/jfugue-chapter2.pdf (from /trunk/src/tonedef/java/jfugue-chapter2.pdf:21)
   A /trunk/src/tonedef/java/Makefile
   D /trunk/src/tonedef/java/jfugue-chapter1.pdf
   D /trunk/src/tonedef/java/jfugue-chapter2.pdf

Relocate pdf files and add Makefile
------------------------------------------------------------------------
r21 | chatura.atapattu | 2010-10-31 15:46:44 -0400 (Sun, 31 Oct 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/java
   A /trunk/src/tonedef/java/HelloWorld.java
   A /trunk/src/tonedef/java/Scale.java
   A /trunk/src/tonedef/java/SmokeOnTheWater.java
   A /trunk/src/tonedef/java/jfugue-4.1.0_20100604.jar
   A /trunk/src/tonedef/java/jfugue-chapter1.pdf
   A /trunk/src/tonedef/java/jfugue-chapter2.pdf

Add sample java programs
------------------------------------------------------------------------
r20 | duanemat | 2010-10-21 14:43:32 -0400 (Thu, 21 Oct 2010) | 7 lines
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/ast.mli
   M /trunk/src/tonedef/parser.mly
   M /trunk/src/tonedef/printer.ml
   A /trunk/src/tonedef/tonedef.ml

Updated program with printing capability.  Current errors generated when:

1.  Parse error with comments written with //.  Will not print comments with /* ... */
2.  Parse error for code outside of functions.  That is probably due to our definition of
 program in AST, which is limited to func list.  I will look into allowing strings.
3.  Does not indent code on print out.  I will fix this, but it would require paying attention
 to whitespace, which isn't really necessary for the code.

Added tonedef.ml, which is needed for the code to compile.  Missed that the first time.  My bad.
------------------------------------------------------------------------
r19 | kramkishun | 2010-10-21 09:51:31 -0400 (Thu, 21 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile

Makefile wasn't working because there's no such thing as tonedef.cmo. File extentions don't
really mean anything on unix based systems but if putting .exe makes it easier for you guys then we can
------------------------------------------------------------------------
r18 | duanemat | 2010-10-20 23:22:12 -0400 (Wed, 20 Oct 2010) | 3 lines
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/printer.ml

So this compiles and will display tokens, though there are some obvious problems that will
need to be hammered out.  I have the Makefile generate a .exe file, but feel free to change if needed.

When compiling with the printer.ml, you need to clean first because of some inherent issues with ocamlc
------------------------------------------------------------------------
r17 | duanemat | 2010-10-20 14:33:58 -0400 (Wed, 20 Oct 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/printer.ml

An incomplete printer for displaying the parsed tokens.  Will hopefully have something up
 later today.  Don't worry about until I post a final version.
------------------------------------------------------------------------
r16 | kramkishun | 2010-10-18 19:38:33 -0400 (Mon, 18 Oct 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/tests

Added a directory for regression tests.
------------------------------------------------------------------------
r15 | kramkishun | 2010-10-17 21:55:06 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/scanner.mll

Rollback changes to revision 13
------------------------------------------------------------------------
r14 | duanemat | 2010-10-17 21:41:01 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   M /trunk/src/tonedef/scanner.mll
```

```
A first shot at the scanner.mll.  I may be way off, but it does compile with the included Makefile
------------------------------------------------------------------------
r13 | kramkishun | 2010-10-17 21:30:53 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/scanner.mll

Fixed the regex for pitch literal
------------------------------------------------------------------------
r12 | kramkishun | 2010-10-17 21:24:07 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   D /trunk/src/tonedef/ast.cmi
   D /trunk/src/tonedef/parser.cmi
   D /trunk/src/tonedef/parser.cmo
   D /trunk/src/tonedef/parser.ml
   D /trunk/src/tonedef/parser.mli
   D /trunk/src/tonedef/scanner.cmi
   D /trunk/src/tonedef/scanner.cmo
   D /trunk/src/tonedef/scanner.ml
   D /trunk/src/tonedef/tonedef

Removed all binaries from source. Lets just keep the source files in here and build on our own machines
------------------------------------------------------------------------
r11 | kramkishun | 2010-10-17 21:20:32 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/scanner.cmo
   M /trunk/src/tonedef/scanner.ml
   M /trunk/src/tonedef/scanner.mll
   M /trunk/src/tonedef/tonedef

Fixed an error in the scanner for string literals
------------------------------------------------------------------------
r10 | kramkishun | 2010-10-17 21:17:06 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/scanner.cmi
   A /trunk/src/tonedef/scanner.cmo
   A /trunk/src/tonedef/scanner.ml
   A /trunk/src/tonedef/scanner.mll
   M /trunk/src/tonedef/tonedef

Threw together the scanner and changed the makefile accordingly. Will try to see if
 I can get the sample code from the proposal to tokenize properly.
------------------------------------------------------------------------
r9 | kramkishun | 2010-10-17 18:54:05 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   M /trunk/src/tonedef/ast.mli

Added a newline at the end of the file that svn was complaining about.
------------------------------------------------------------------------
r7 | kramkishun | 2010-10-17 13:06:36 -0400 (Sun, 17 Oct 2010) | 1 line
Changed paths:
   A /trunk/src/tonedef/Makefile
   A /trunk/src/tonedef/tonedef
```

```
Added a makefile since this looks like it will be the main folder we're gonna work out of.
'make' and 'make clean' work. I'll keep the makefile up to date as more files are added.
------------------------------------------------------------------------
r6 | cah2196 | 2010-10-17 12:42:17 -0400 (Sun, 17 Oct 2010) | 3 lines
Changed paths:
   A /trunk/src/tonedef
   A /trunk/src/tonedef/ast.cmi
   A /trunk/src/tonedef/ast.mli
   A /trunk/src/tonedef/parser.cmi
   A /trunk/src/tonedef/parser.cmo
   A /trunk/src/tonedef/parser.ml
   A /trunk/src/tonedef/parser.mli
   A /trunk/src/tonedef/parser.mly

First swing at the ast.mli and parser.mly
Ocamlyacc executes cleanly on the parser - but it's probably not complete
No scanner yet to actually test out the AST constructions
------------------------------------------------------------------------
```