ANIMO

Character Animation Language

Alexei Masterov
Natasha Shamis
Joshua Poritz

Table of Contents

Chapter 1

Introduction

1.1 Overview

1.1.1 Summary

ANIMO is a language that is designed to control an animated character. It allows a programmer to compose instructions specifying a series of body movements, enabling a character to perform a variety of motions. ANIMO has an easy-to-use syntax, combining numerous features characteristic of higher-level languages with unique functions of its own.

1.1.2 Applications

There are three major domains in which our language would likely be employed:

a. Computer Animation
Computer animation is used very widely in the movie industry. ANIMO may become a standard for animators and an alternative for visual interfaces because it allows the reuse of code while providing the flexibility by means of parameters.

b. Computer Games
Computer games and virtual environments is another domain where our language can be used by both programmers and players. In places like the Metaverse (http://en.wikipedia.org/wiki/Snow_Crash), users can "learn" to dance by writing an ANIMO program and use it to impress their friends. The creators of such environments can use ANIMO for animating crowds by re-using the same code with different parameters.

c. Robot Control
Finally, ANIMO can be used in a real world for operating toy robots. It should be relatively easy to port it to allow control for a humanoid robot that has a set of joints arranged in a hierarchical fashion.

1.1.3 BVH File Format

The BVH file format was originally developed by Biovision, a motion capture services company, as a way to provide motion capture data to their customers. The name BVH stands for Biovision hierarchical data.
It contains the hierarchical structure of joints and end effectors, the motion capture data as a list of figure coordinates in space, and a list of joint rotation angles over a sequence of frames.

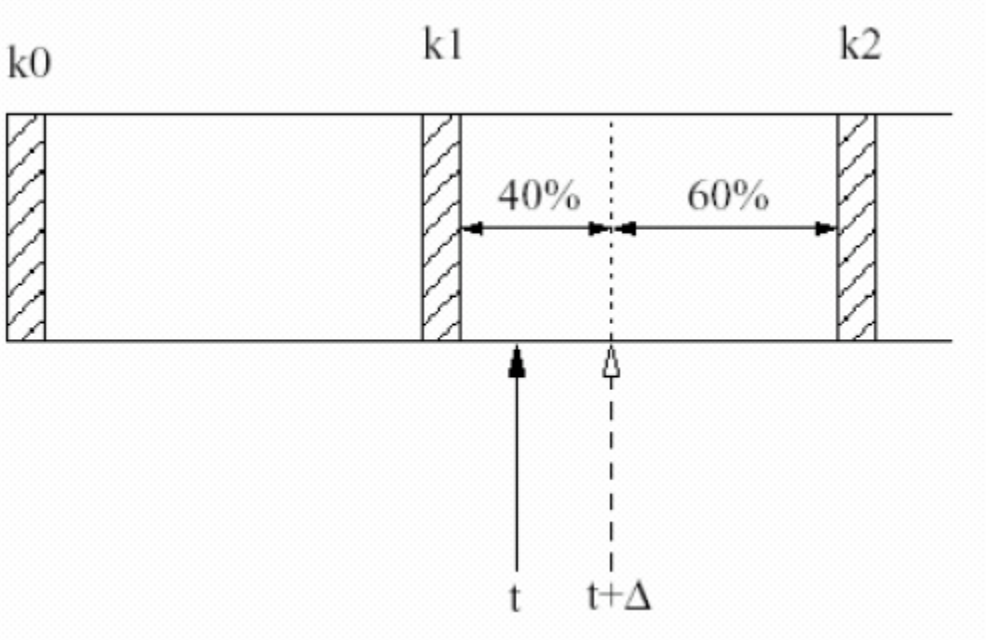
Sample BVH file:

```
HIERARCHY
ROOT Hips
{
 OFFSET 0.00 0.00 0.00
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
   JOINT Chest
   {
    OFFSET 0.00 5.21 0.00
     CHANNELS 3 Zrotation Xrotation Yrotation
      …
   }
}
MOTION
Frames: 2
Frame Time: 0.033333
8.03 35.01 88.36 -3.41 14.78 -164.35 13.09 40.30 -24.60 …
7.81 35.10 86.47 -3.78 12.94 -166.97 12.64 42.57 -22.34 …
```

The first three columns correspond to the X, Y, and Z coordinate of the root joint position in space, and the rest of the columns define the local rotation angle of each joint in the hierarchy along the X, Y, and Z-axis. Joints are rigidly attached; therefore, they can only rotate, and not move, in space.

If one rotates the joint in the hierarchy, all the joints in the hierarchy below get rotated with it:

The “Frame time” parameter specifies the time interval between frames. The rendering software then interpolates all the joint positions in between frames.



So how do we make the figure move in a consistent fashion? The hierarchy in the BVH file specifies the initial position of the figure (k0). Then each row specifies what will happen to the figure at the next time interval (k1, k2, …, kN). Each movement is a combination of joint rotations. In our language, we allow the user to specify each joint rotation individually, indicating the time at which the rotation should begin and the duration for which it lasts; this allows several movements to be performed simultaneously. This operation of combining different movements is called “blending”. By blending different joint rotations, the user can make the figure perform arbitrary complex movements through space.

1.1.4 Compiler: Inputs and Outputs

Motion will be simulated through the use of BVH files, which contain hierarchical information for the skeletal structure of the animated characters. Our compiler will accept an ANIMO file(s) and a BVH file (or still BVH file) containing the character’s initial position, and output a new BVH file containing a set of instructions in the form of joint movements for the character.

1.2 Language Elements

1.2.1 Data Types, Operators, Loops, and Conditionals

ANIMO uses conditionals such as “if” and loops such as “for”. Since all of our data represents joint rotations, we limited our data types to doubles and have addition,

subtraction, division, multiplication, and modulus operators defined for them with standard (mathematical) precedence.

1.2.2 Basic Commands: rotate() and move()

The most basic commands in ANIMO's default library are:

```
rotate(joint, start_time, duration, x, y, z);
```

(which maps nicely to the 3 columns that correspond to a given joint rotation)

```
move(start_time, duration, x, y, z);
```

(which maps to the figure's position in space).

"rotate" and "move" comprise ANIMO's smallest building blocks. A programmer may combine these fundamental motions to create more complex movements.

1.2.3 Blending

One may say that "rotate" and "move" are such primitive commands that they provide no ostensible benefit over editing BVH files directly. Our compiler, however, provides the capability of blending several movements together to create fairly complex concurrent motions. These more complex motions, in turn, may be blended to form even more sophisticated character movements.

1.2.4 User-Defined Functions

Functions are fundamental to the purpose of our language. A function encapsulates a series of concurrent and sequential movements into a named block, such as `kneel`(), which may be called by any other function that seeks to perform the motion with that name. A program to make a skeleton jump, for example, may contain a main routine consisting solely of four function calls in succession — `kneel(), straighten(), elevate(), and descend()` — which results in a much more readable program than several dozen unnamed lines specifying the same joint transformations.

1.3 Examples

1.3.1 Sample User-Defined Functions

Below is an example of a function that would move a character's left leg:

```
left_leg(..)
{
      rotate(LeftHip, ..);
      rotate(LeftKnee, ..);
      rotate(LeftAnkle, ..);
```

```
        move(..);
        rotate(LeftHip, ..);
        rotate(LeftKnee, ..);
        rotate(LeftAnkle, ..);
        move(..);
        rotate(LeftHip, ..);
        rotate(LeftKnee, ..);
        rotate(LeftAnkle, ..);
        move(..);
        rotate(LeftHip, ..);
        rotate(LeftKnee, ..);
        rotate(LeftAnkle, ..);
        move(..);
}
```

Or, equivalently:

```
left_leg(..)
{
        for (i = 0 to 4)
        {
                rotate(LeftHip, ..);
                rotate(LeftKnee, ..);
                rotate(LeftAnkle, ..);
                move(..);
        }
}
```

By blending different functions together, users may create more complex movements:

```
step_right(..)
{
        left_hand(..);
        right_leg(..);
        move(x, y);
}
step_left(..)
{
        right_hand(..);
        left_leg(..);
        move(x, y);
}
```

1.3.2 Sample Program

```
march()
{
        for (I = 0 to 10)
        {
                if (I % 2 = 0)
                        if (I % 4 = 0)
                        {
                                step_right(..);
                                wave_right();
                        }
```

```
                        else
                              step_right(..);
                  else
                        step_left(..);
            }
      }
```

REFERENCES:
[1] http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html
[2] “Design and implementation of real-time character animation library”. Hakan Almer, Eric Erlandson.

Chapter 2

Language Tutorial

2.1 Writing an ANIMO Program

2.1.1 BVH Files

The first thing you need to do when writing an ANIMO program is include the BVH (or still BVH) file that the program is going to work with. For example:

```
include "march.sbvh";
```

The SBVH file contains the hierarchical structure of the joints of an animated character and the character's initial position. Once it is included, you can use the internal functions available to you or create your own, more complex functions, in order to manipulate the character. A BVH file already contains pre-existing motions, so any new movements you include will simply change the already existing movements.

2.1.2 Internal Functions

ANIMO contains the internal functions *move* and *rotate*. In order to rotate a joint, specify the x, y, and z coordinates that you would like to rotate the joint to, and the time interval over which this rotation should take place. For example, to rotate the character's head to position *x*, *y*, *z*, starting at *start_time* and continuing for *duration* number of seconds:

```
rotate(Head, start_time, duration, x, y, z);
```

To move the entire character (from its center) to position *x*, *y*, *z*, over the time interval from *start_time* to *start_time* + *duration*:

```
move(start_time, duration, x, y, z);
```

In order to set the start time as the current time, use the *curtime()* internal function, which returns the latest time at which any animation has been performed. Upon calling either the *rotate()* or *move()* functions, the timeline manipulation is internally converted to frames.

A couple of useful functions (mainly for debugging purposes) are *print()* and *what()*. *print()* takes a list of variables, and writes their names and values to `stdout`. *what()*, similarly, takes a list of variables, and writes their types, as well as their names and values to `stdout`. These functions can only be used on BVH joints and floating points; for floating points, only the value is written. Bonus: ANIMO also contains various internal functions that automatically perform common mathematical evaluations for you, but we'll leave those for you to explore!

2.1.3 User-Defined Functions

When declaring a new function, the function name should be preceded by the reserved word, *func*, and followed by the function arguments. The body of the function must be encapsulated in braces. For example, to create a function called *left_ankle*, which accepts arguments *x*, *a*, *b*, and *c*, and rotates the character's left ankle:

```
func left_ankle(x, a, b, c)
{
      rotate(LeftAnkle, x, 1, a, b, c);
}
```

To create a more complex movement, such as *left_step*, which enables the character to take a step with the left leg and requires the movement of several joints:

```
func left_step ( begin_frame, end_frame )
{
      left_hip(x, a, b, c);
      left_knee(x, a, b, c);
      left_ankle(x, a, b, c);

}
```

In turn, even more complex functions can be built via calls to simpler functions.

2.1.4 Loops and Conditionals

The *for* loop and *if* conditional have the same functionality as in most other programming languages. The syntax for the *for* loop is the following:

```
for (k = 1 to 4)
```

The syntax for the *if* conditional is the following:

```
if (j == 70)
```

Loops, functions, and conditionals can all be nested.

2.2 Compiling an ANIMO Program

Once you have completed your ANIMO program, run the following code on the command line in order to compile your code:

```
java AnimoMain <input ANIMO filename>
```

This creates a new BVH file, containing the updated character joint movements. The default output file is *out.bvh*, but if you would like to specify a filename, use the following format:

```
java AnimoMain <input ANIMO filename> [-o <output BVH filename>]
```

The last compilation feature we have is verbose mode, which is very helpful in debugging. Using verbose mood, you will be able to view the structure of the syntax tree, the program output, a stack trace (so that you can see specifics on where in the compiler an error was generated), and a list of existing global variables and symbol tables. To compile in verbose mode, use the following syntax:

```
java AnimoMain <input ANIMO filename> [-o <output BVH filename>][-v]
```

2.3 Running an ANIMO Program

Run your new BVH file through a BVH player, and enjoy the results!

Chapter 3

Language Reference Manual

3.1 Introduction

This manual describes the ANIMO language as decided by the ANIMO team on the Twentieth of December, 2005. The manual serves as the authoritative standard for the ANIMO language and delineates all of ANIMO's rules, conventions, and built-in features.

3.2 Lexical Conventions

A program consists of one or more translation units stored in files. It is translated in two phases. The first phase attempts to interpolate (recursively) all files specified in `include` statements. These interpolated files may be ANIMO programs or "still BVH" (hereafter abbreviated SVBH) files that the program will animate. The second phase compiles the single resulting file into an animated BVH file, which is written to disk.

For a formal description of the SBVH format, see (http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html). The rest of this section describes the tokens that comprise an ANIMO source file.

3.2.1 Tokens

There are six classes of tokens: identifiers, keywords and built-in functions, constants, string literals, operators, and other separators. Blanks, tabs, carriage returns, newlines, and comments as described below (collectively "white space") are ignored except as separate tokens. Some white space is required to separate adjacent identifiers, keywords, and constants.

3.2.2 Comments

ANIMO allows two styles of comments. Single-line comments, which are introduced by a double slash ("//"), cause the compiler to ignore the remainder of that line. Multi-line comments begin with a slash followed by a star ("/*") and end with the first subsequent instance of a star, followed by a slash ("*/"); the compiler ignores all text in between those two delimiters. Comments do not nest, and they do not occur within string literals.

3.2.3 Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore ("_") counts as a letter. Identifiers are case sensitive. All variables, BVH joint names, and function names fall under the classification of identifier.

3.2.4 Keywords and Built-in Functions

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
else        include     return
for         func        to
if
```

The following identifiers are reserved as names for built-in functions, and no user-defined function may have these names:

```
print       what        move
rotate      curtime     floor
ceil        sin         cos
tan         sqrt
```

3.2.5 Constants

Almost exclusively, the only constant used throughout an ANIMO program is the floating point. A floating constant consists of an integer part, a decimal part, a fraction part, an e or E, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing.

For convenience, the following mathematical floating point constants are built into ANIMO. They have global scope, and their values may be changed or overwritten in an ANIMO program at any time:

```
PI    E
```

The only other constant employed in an ANIMO program is the string literal, which has its place only as the parameter to an `include` statement. A string literal consists of a sequence of characters enclosed in double quotes, as in "... ". Consecutive pairs of double quotes within a string literal are reduced to one double quote and that double quote is interpreted as part of the string literal, not as its end delimiter.

A special constant, with identifier NULL and type “unknown,” is returned by user-defined function with no explicit return value and built-in functions `print()`, `rotate()`, `move()`, and `what()`. The NULL identifier may not be used in any expression, assignment, or function call.

3.3 Expressions

This section places each type of expression operator in a separate subsection in order of precedence, from highest to lowest. The operators within each subsection have the same precedence. Left- or right-associativity is specified in each subsection for the operators discussed therein.

3.3.1 Primary Expressions

Primary expressions group left-to-right, and include identifiers, constants, parenthesized expressions, and function calls. The type and value of a parenthesized expression are identical to those of the simple expression. The presence of parentheses does not affect whether the expression is an l_value. A function call is composed of a primary expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the arguments to the function.

3.3.2 Unary Operators

The unary operators, +, -, ++, --, and ! group right-to-left. They yield the positive, negative, increment, decrement, and negation of the expressions, correspondingly.

3.3.3 Multiplicative Operators

The multiplicative operators *, /, and % group left-to-right. The binary * operator indicates multiplication. The binary / operator indicates division. The binary % operator yields the remainder from the division of the first expression by the second.

3.3.4 Additive Operators

The additive operators + and – group left-to-right. The + operator yields the sum of the expressions, and the – operator yields the difference of the expressions.

3.3.5 Relational and Equality Operators

The relational operators <, >, <=, and >= group left-to-right. The equality operators == and != can be grouped in either direction. Both the relational and the equality operators yield 0 if the specified relation is false and 1 if it is true.

3.3.6 && Operator

The && operator groups left-to-right. It returns 1 if both expressions are non-zero, 0 otherwise. If the first expression evaluates to 0, the second expression is not evaluated.

3.3.7 || Operator

The || operator groups left-to-right. It returns 1 if either of its expressions is non-zero, 0 otherwise. If the first expression is non-zero, the second expression is not evaluated.

3.3.8 Assignment Operators

The assignment operators =, +=, -=, *=, /=, and %= group right-to-left. They require an l_value as their left operand, and the type of an assignment expression is the same as the left operand. After the assignment takes place, the resulting value is stored in the left operand.

3.4 Functions

A function encapsulates a sequence of ANIMO statements into a named piece of code. All functions in an ANIMO source program consist of a declaration, followed immediately by zero or more statements enclosed in curly braces. A declaration has the form:

func_def:
`"func"` *identifier "(" var_list ? ")"*
var_list:
*identifier ("," identifier)**

The return value of a function is always a floating point constant. If no specified value is returned from within the function, the return value defaults to 0.

3.5 Statements

Except as indicated, statements are executed in sequence.

3.5.1 Expression Statement

Most statements are expression statements, which have the form:

expression ;

Usually, expression statements are assignments or function calls.

3.5.2 Compound Statement

So that several statements can be used where one is expected, the compound statement is provided:

compound-statement:
{ statementlist }

statementlist:
statement
statement statement-list

3.5.3 Conditional Statement

The two forms of the conditional statement are:

`if` *(expression) statement*
`if` *(expression) statement* `else` *statement*

In both cases, the expression is evaluated and if it is nonzero, the first sub-statement is executed. In the second case, the second sub-statement is executed if the expression is 0. As usual, the ‘‘else’’ ambiguity is resolved by connecting an else with the last encountered else-less `if`.

3.5.4 For Statement

The `for` statement has the form:

`for` *(id expression “to” expression) statement*

3.5.5 Return Statement

A function returns to its caller by means of the `return` statement, which has one of the following forms:

`return` *;*
`return` *(expression) ;*

In the second case, the value of *expression* is returned to the caller of the function. In the first case, a NULL identifier is returned (see section 3.2.5). Flowing of the end of a function can result in either case: of no assignment occurred in the outermost block of the function, the NULL identifier is returned; otherwise, the function’s return value is equal to the l_value of the last assignment that occurred in the outermost block of the function.

3.5.6 Include Statement

The `include` statement is used to import BVH (and SBVH) files and other ANIMO files.
Note: Only one BVH file can be imported per program, and the same ANIMO file cannot be imported twice. The files must be imported before any statements are executed.

The `include` statement has the following format:

`include` <*filename*> *;*

If the filename ends with “.bvh” or “.sbvh,” the BVH parser becomes invokes to parse the file, and all joint names in the skeleton become stored in the BVH joints symbol table.

3.6 Scope

An ANIMO program has two levels of scope: global scope, and block scope. An identifier initialized outside of any block (a group of statements enclosed in curly braces) falls under global scope and is visible throughout the remainder of the source file. By contrast, an identifier initialized within a block is restricted to block scope and is visible only for the remainder of the block in which it is declared.

BVH joint names are the exception to normal scoping rules. Any BVH joint name, including those declared within curly braces (i.e. not at the very top of the joint hierarchy), automatically has global scope and may be passed to any function for the remainder of the source file after which it appears.

As expected, function identifiers have global scope, whereas parameters specified in variable lists within function declarations have block scope. A function identifier cannot be used more than once since ANIMO does not provide for overloading.

The namespace for functions is independent from the namespace for BVH joints and floating points; as such, the same identifier could refer to both a function and either a BVH joint or floating point. The latter two types, though they are internally stored in different symbol tables, artificially occupy the same namespace because ANIMO disallows a BVH joint and a floating to be bound to the same identifying name. A variable, once declared as a floating point or BVH joint, may not be assigned to a variable of the other type until it goes out of scope.

Variables assigned to BVH joint names are called BVH joint *aliases* and may be used in all contexts that BVH joint names can; aliases, therefore, are stored in the same global table as BVH joint names and reside in global scope. Unlike the actual BVH joint names, however, a single alias may be assigned to different BVH joints throughout its lifetime (but never to a floating point).

Chapter 4

Project Plan

4.1 Team Responsibilities

Over the course of the semester, our team met at least once every week, in order to discuss the status of our project and ask our TA questions. At the beginning of the term, when we were still figuring out the course our project should take, and at the end of the term, when it was crunch time, we met several times a week, in order to have all our heads working at the same time, and to smooth over each other's misunderstandings. We also wrote periodic e-mails to the team, updating each other on changes that have been made, many times every week.

Each member of the team had an approximately equal part in the coding, and we reviewed each other's code to check for consistency, sometimes suggesting alternate implementation schemes. We did not establish concrete goals for each member right away, but rather split up the work as we went along. Thus, the work was divided in the following manner, with constant interaction and improvement by all team members:

Alexei Masterov
- team leader, created project idea
- research on animation and motion
- features for working with BVH files
- matrix implementation
- sample programs for testing purposes

Natasha Shamis
- parser, walker
- symbol table implementation
- compiler implementation
- documentation

Josh Poritz
- lexer, parser, walker
- data type implementation
- function implementation
- testing

4.2 Project Timeline

The following is our list of completion dates for project milestones. See section 4.5 for a more detailed project log.

Project idea defined	Sept 15
Proposal	Sept 27
Grammar defined	Oct 18
Language Reference Manual	Oct 19
Supporting Java classes	Nov 17
Walker defined	Nov 21
Testing complete	Dec 16
Final report	Dec 20

4.3 Software Development Environment

The code for the compiler was written in Java 2 SE 1.5. The code for the lexer, parser, and walker was written using Antlr. However, most of the walker functionality was written in the supporting Java classes, and the walker was just used to call instances of the classes. We used CVS for version control and source code management.

4.4 Programming Style Guide

Every so often, we made changes to each other's code in order to be more consistent with each other's programming styles. Although our styles are different in minor ways, we will attempt to outline them here.

4.4.1 Java Coding Style

All class names begin with "Animo" followed by the class description, with each word capitalized.

In Josh and Natasha's code, a left brace is at the end of the previous line (with a space before it), and the right brace has its own line and is placed in the same column as the first character of the line before the left brace. There are no spaces between arguments and their parentheses (except in the walker) or between function names and their parentheses. Indentations are usually about three or four spaces, and tabs are rarely used.

Alexei's code has a space between arguments and their parentheses and places left braces on a new line aligned with the first character of the line before the left brace. He uses tabs for indentations. (Perhaps Columbia undergraduate courses are responsible for Josh and Natasha's similar styles.)

Overall, all three team members tried to place as little code as possible on the same line (for example, an "else" statement is always on a separate line from an "if" statement) to avoid confusion.

Variable names are short and lowercase. Method names are composed of English words, the first characters of which are capital, except for the first word in the name. We tried to comment our code as often as possible.

4.4.2 Antlr Coding Style

If an Antlr rule is short and contains only one choice without any action, it is written in one line. The colon is always placed in the ninth column unless the string in before it is too long. In the one line mode, the semicolon is at the end of the line. If an Antlr rule has multiple choices, the semicolon is placed in a separate line, in the same column as the colon. Short actions are on the same line as their rule, while long actions are on the line after the rule. Token names are in uppercase, and non-terminals are in lowercase, with an underscore to separate words.

4.5 Project Log

This is a more detailed record of our progress throughout the semester.

Each team member comes up with language idea	Sept 8
Research done on animation and BVH libraries	Sept 13
Project idea defined	Sept 15
Language details defined	Sept 22
Draft of language proposal	Sept 25
Language proposal complete	Sept 27
Code overview developed	Oct 2
Lexer written	Oct 10
Parser written	Oct 12
Draft of Language Reference Manual	Oct 16
Grammar complete	Oct 18
Language Reference Manual complete	Oct 19
Back-end Java classes complete	Nov 9
Front-end Java classes complete	Nov 17
Walker defined	Nov 21
Revising of Java classes complete	Dec 11
All sample programs written	Dec 15
Testing complete	Dec 16
Draft of final report	Dec 17
Final report complete	Dec 20

Chapter 5

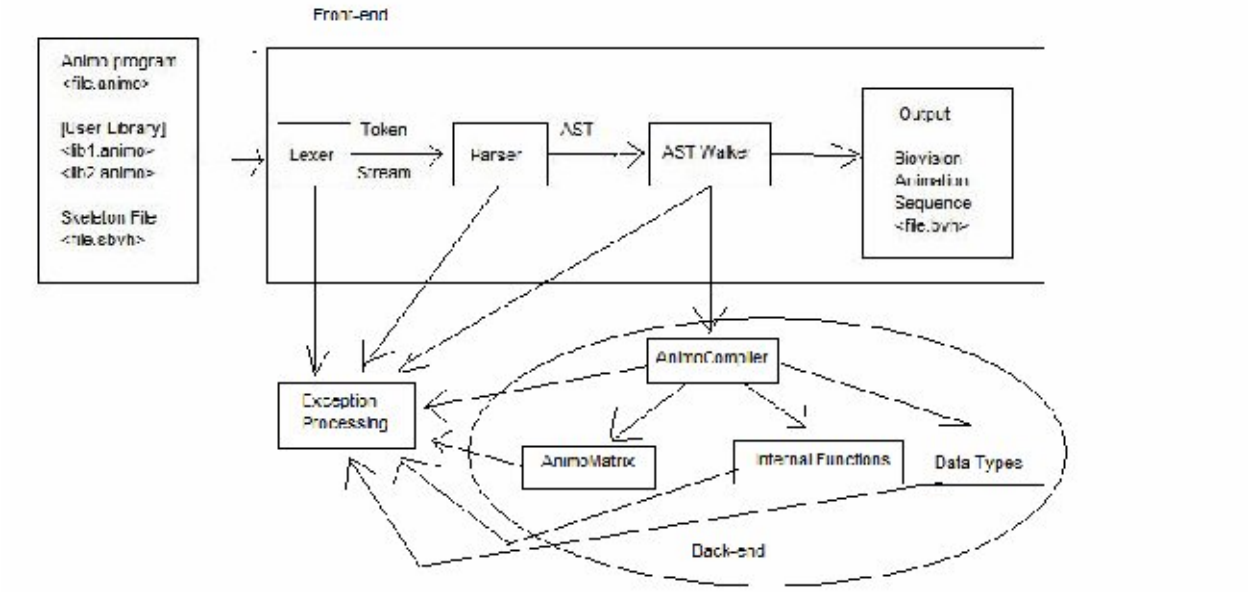
Architectural Design

5.1 Compiler Structure

The ANIMO compiler consists of:

- a lexer that converts the input into tokens
- a parser that analyzes the syntactic structure of the program and converts it to an abstract syntax tree
- a tree walker that traverses the tree and calls underlying functions
- an AnimoCompiler class that manipulates symbol tables and data types and performs all functionalities for the walker
- an AnimoMatrix class that manipulates the motion matrices of BVH files
- an exception processing mechanism

Block diagram:



5.2 Front-end

The lexer, parser, and tree walker are implemented by Antlr. The lexer and parser are implemented in Antlr file *grammar.g*, and the walker is implemented in Antlr file *walker.g*. Source file *walker.g* contains calls to the methods in class *AnimoCompiler*.

The *main()* method is in the class *AnimoMain*. It accepts a file, and then initializes the lexer, parser, and tree walker. *AnimoMain* also handles errors and exceptions that occur in other classes, printing corresponding messages.

5.3 Back-end

The class *AnimoDataType* represents a generic data type. All objects stored in the symbol tables of an ANIMO program inherit from this class and are cast up to *AnimoDataType* before being stored in the tables. Floating points, functions, and BVH joints are derived from this class, and their functionalities are stored in *AnimoFloatingPoint*, *AnimoFunction*, and *AnimoBVHJoint* classes. The class *AnimoFunction* represents any user-defined or internal function. Internal functions have unique identifiers and are implemented in *AnimoInternalFunction*, a subclass of *AnimoFunction*.

The *AnimoException* class handles exceptions, allowing them to be created and thrown.

When the *AnimoCompiler* class for an ANIMO program is instantiated, three global symbol tables are set up – one for functions, one for BVH joints, and one for floating points. The table for floating points is the outermost parent in a stack, enabling multiple scopes. When searching for a key, the compiler first searches the BVH joint symbol table, and then the floating point symbol table. (The function symbol table is only searched when encountering invocation.) A joint and a floating point cannot have the same identifier. Symbol table functionality is implemented in the *AnimoSymbolTable* class.

Actual modification of the input occurs in the *AnimoMatrix* class. This class appends a motion matrix to the input file if it is an SBVH file, or manipulates the existing motion matrix in the file if it is a BVH file. When the *move()* or *rotate()* functions are called, the rotation values in the appropriate columns of the motion matrix are changed. To prevent the rest of the values in the matrix from reverting back to 0 once the movement is complete (hence, returning the joint to its original position), the last rotation values are repeated until the next movement is processed or until the end of the matrix if there are no more movements.

5.4 Implementation

As mentioned in section 4.1, Joshua and Natasha implemented the front-end. The back-end was split between the three of them, with Joshua implementing the *AnimoFunction*, *AnimoDataType*, and *AnimoException* classes (and their subclasses), Natasha implementing the *AnimoCompiler* and *AnimoSymbolTable* classes, and Alexei implementing the *AnimoMatrix* class and various features for working with BVH files.

Chapter 6

Testing Plan

In order to test various features of our language, Joshua, who was responsible for testing our compiler, wrote several ANIMO programs, each of which tested a particular feature. There were seven tests in total, which tested proper usage of: *for* loops, *if* conditionals, user-defined functions, built-in functions, operators, scoping, and the *rotate* and *move* internal functions in particular. ANIMO comments were successfully used throughout. For each feature, Joshua tested every possibility he could conceive of, including those actions that were legal in ANIMO and those that weren't. When he, as the user, wrote a segment of code that was illegal, he made sure that our compiler did the proper error checking and displayed adequate messages as to their source. As an example, here are two tests that were conducted. The others can be found in the Appendix.

6.1 Testing Scoping

```
include "test.sbvh";

// Use of floating point declared outside a block inside the block
a = 5;
if (a == 5)
    rotate(Neck, 0, 1, a, a, a);

// Changing the floating point, then re-using it
a = 6;
rotate(Neck, 1, 1, a, a, a);

// Changing a floating point inside a block
a = 8;
if (a == 8)
{
    a = 7;
    rotate(Neck, 2, 1, a, a, a);
}
rotate(Neck, 3, 1, a, a, a);

// Using a floating point declared inside a block outside the block
/*
if (1)
{
    rotate(Neck, 4, 1, a + 1, a + 1, a + 1);
    b = 10;
}
rotate(Neck, 5, 1, b, b, b);
*/

// Use of for loop counter
for (cnt = 11 to 14)
```

```
    rotate(Neck, cnt, 1, cnt, cnt, cnt);
/* rotate(Neck, cnt + 1, 1, cnt + 1, cnt + 1, cnt + 1); */

// Demonstrate that aliases for BVH joints are global
if (1)
    n = Neck;
rotate(n, 15, 1, 15, 15, 15);

// Changing the variable type (should be disallowed)
/*
n = 16;
rotate(Neck, n, 1, n, n, n);
m = 15;
m = Neck;
*/

// More complex example testing scope pretty comprehensively
func a(x)
{
   return x;
}
/* blah = x; */  /* Trying to use variable after its scope */
func b(y)
{
    p = 23;
    move(y, 1, y, y, y);
    y = 20;
    for (i = a(6) + 10 to a(7) + 10)
    {
        j = Head;
        rotate(Neck, i, 1, i, i, i);
        if (1)
          y = 19;
      rotate(Neck, 21 + i - 16, 1, y, 21, i);
    }
    /* blah = i; */  /* Trying to use loop counter after its scope */
    rotate(j, p, 1, p, p, y);
}
b(24);
```

As in any ANIMO program, the first thing that is done is the inclusion of a still BVH file. Then, the program makes attempts to use and re-use floating points and BVH joints outside of and within their scopes. For example, if a floating point is declared inside a block, it will not be valid outside of the block. When conducting this test, the ANIMO compiler threw an exception, showing the user that this was an invalid operation. Other tests that were conducted involved changing a variable's type (which is not allowed in our language), using an alias for a BVH joint outside of the block where it was declared (permissible because aliases for BVH joints are global), and various combinations of these tests. All the illegal tests that successfully threw exceptions were commented out so that they are retained without crashing the program. This enables the program to successfully compile and output a sample BVH file, a portion of which is shown below. The entire file is too large to include here, but you can see that a motion matrix has been created and that the values of the rotations are gradually being changed within the frames.

```
HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
      …

  }
}
MOTION
Frames: 98
Frame Time: 1.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 10.0000 10.0000 10.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 3.0000 3.0000 3.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 11.0000 11.0000 11.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 4.0000 4.0000 4.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 12.0000 12.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 5.0000 5.0000 5.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 13.0000 13.0000 13.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 14.0000 14.0000 14.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
…
```

6.2 Testing *Move* and *Rotate*

```
include "rotMovTest.sbvh";

// rotate and move commands with whole number start time and duration
arguments
```

```
rotate(LeftKnee, 0, 4, 30, 45, 60);
rotate(LeftWrist, 0, 1, 5, 10, 10);
move(2, 3, 70, 20, 5);

// Rotation/movement with negative and fractional (x, y, z)
rotate(LeftKnee, 4, 4, -30, -45, -60);
rotate(LeftWrist, 1, 1, -5, -10.5, -10.5);
move(5, 3, -70, -20.5, -5.5);

// rotate and move commands with fractional start time and/or duration
for (i = 1 to 10)
    rotate(RightElbow, curtime(), 0.15 * i, i * 2, i * 5, i * 10);
for (i = 1 to 10)
    move(curtime(), 0.15 * i, i * 2, i * 5, i * 10);
x = ceil(curtime());
rotate(LeftShoulder, x + 0.2, 0.2, 25, 25, 25);
rotate(LeftShoulder, x + 0.4, 0.6, 30, 30, 30);

// rotates and moves overlapping some frames
rotate(RightShoulder, 15, 5, 64, 64, 64);
rotate(RightShoulder, 18, 5, 73, 73, 73);
move(15, 5, 64, 64, 64);
move(18, 5, 73, 73, 73);

// rotates and moves COMPLETELY overlapping the same frames
rotate(LeftCollar, 25, 5, 80, 80, 80);
rotate(LeftCollar, 25, 5, 90, 90, 90);
move(25, 5, 80, 80, 80);
move(25, 5, 90, 90, 90);

// rotate with invalid joint name
/*
rotate(LeftNeck, 42, 1, 60, 60, 60);
*/

// rotates and moves with zero durations
/*
rotate(Neck, 42, 0, 60, 60, 60);
move(42, 0, 60, 60, 60);
*/

// rotates and moves with negative durations and start times
/*
rotate(Neck, 42, -1, 60, 60, 60);
move(Neck, -1, 60, 60, 60);
rotate(Neck, -0.5, 1, 60, 60, 60);
move(-0.5, 1, 60, 60, 60);
*/

// rotates and moves with wrong numbers of parameters
/*
rotate();
rotate(Neck);
rotate(Neck, 4, 7);
rotate(Neck, 4, 7, 0, 0, 0, 0);
move();
move(4, 7);
```

```
move(4, 7, 0, 0, 0, 0);
*/

// rotates and moves with invalid parameters
/*
a = 3;
rotate(5, 10, 1, 0, 0, 0);
rotate(a, 10, 1, 0, 0, 0);
rotate(Neck, Neck, 1, 0, 0, 0);
rotate(Neck, 10, Neck, 0, 0, 0);
rotate(Neck, 10, 1, Neck, 0, 0);
rotate(Neck, 10, 1, 0, Neck, 0);
rotate(Neck, 10, 1, 0, 0, Neck);
move(Neck, 1, 0, 0, 0);
move(10, Neck, 0, 0, 0);
move(10, 1, Neck, 0, 0);
move(10, 1, 0, Neck, 0);
move(10, 1, 0, 0, Neck);
*/

// scheduling a rotation way ahead of time
rotate(RightCollar, 50, 10, 120, 120, 120);

// scheduling another rotation in between
rotate(RightCollar, 35, 5, 40, 40, 40);
```

Once again, any tests that were unsuccessful (as required) were commented out to allow the program to compile. Both functions were tested with whole, fractional, and negative rotation values to make sure that all joint rotations were possible. But more importantly, this program included tests for concurrency of movements. Since movements and rotations take place over a time interval, there should be no problem with several different movements happening at the same time.

We also conducted a test to ensure that ANIMO properly interpolates movements. A simple BVH player does not provide an interpolation feature, and ANIMO's implementation of it results in smooth animated movements on virtually any BVH player! To test this, we simply changed the frame time in the output BVH file. When the frame time increases, the movements become sharper, and when the frame time decreases, the movements become smoother.

Chapter 7

Lessons Learned

7.1 Natasha's Learning Experience

Before beginning to work on any part of the project, outline what needs to be done and approximate how much time it will take. This will result in much more efficient planning and implementation. Try to set realistic estimates on time management and strive to meet your own deadlines. I found that I sometimes put something off because I expected it to take a long time and didn't have the opportunity to implement it immediately. Fifteen minutes after beginning to code, I realized that I was almost done with the section and completely overestimated the time required to implement it. Since finishing that section of the project sooner would have allowed my teammates to work on other sections (that relied on mine), it would have been more efficient if I initially reviewed what needed to be done in more detail, resulting in a more accurate time estimate. Conversely, we sometimes left too *little* time for an implementation that took *longer* than we planned, resulting in unnecessary rushing at the end.

7.2 Alexei's Learning Experience

Most of what I now know about animation I learned while creating the ANIMO language. Previously having a vague understanding of BVH files and players, I learned a lot more about the software and hardware used for motion capture, the file formats used in storing motion data, and systems that are used for manipulating, editing, and playing motion cap data. Through the comparison of various players, I learned to appreciate different features, such as the OSL player, which internally smoothes motions, as opposed to a simple BVH player, which does not. I also learned that most animated motion that appears smooth and lifelike is created by digitally videotaping a person who is clothed in special gear and performs the required movements. This is an expensive and inefficient method, which is difficult to reproduce. And here, the gateway is opened for a language such as ANIMO.

7.3 Joshua's Learning Experience

In approaching the design of our compiler, we started with the source code for the 2003 Mx project and modified it to suit the specifications defined in our LRM. In the end, we practically rewrote the entire Mx compiler, and one would have a hard time figuring out that we started with Mx as our base by looking at our source code. It turned out that around 80% of our coding time was spent figuring out Mx rather than writing our own source code. In retrospect, it would have been more sensible and efficient to simply write our own compiler from scratch.

Furthermore, some of Mx's limitations manifested themselves in our compiler and complicated

testing a little bit. For example, Mx does not have the debugging option that causes runtime errors to print an error message and continue; rather, an exception gets thrown and the program terminates. Adding this option would have entailed enclosing all those exception-throwing statements in an "if" block which tests whether that option was set, and we found the overhead and messiness of this change to be unreasonable. Because we lacked this option, however, I had to write and run bad code segments testing ANIMO compilation errors one segment at a
time; I couldn't write a single program testing all the bad cases like I wanted to, which was sort of a nuisance.

7.4 Advice for Future Teams

Consider different methods of implementing your ideas. Do not stop on the first method that you have thought up because there may be alternate ways to do something that end up being simpler or more efficient. For example, before deciding to implement interpolation through a timeline scheme, enabling several movements to be executed at once based on start time and duration, we were going to implement interpolation by overloading the “+” operator to connect different movements. Although we were discussing the latter implementation over the course of several weeks, we later began to consider the former method and realized the implementation was simpler and more intuitive. A good way to pool as many alternatives as possible is to have each member of the team come up with his/her own idea, be it for the project proposal or for the actual implementation. That way, several different options can be considered, and the team can work together to find the best one.

Appendix: Complete Code Listing

```
/***********************************************************
 * grammar.g      Authors: Joshua Poritz, Natasha Shamis
 ***********************************************************/

class AnimoAntlrLexer extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = AnimoAntlr;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT   :   '0'..'9';

WS      : (' ' | '\t')+          { $setType(Token.SKIP); }
        ;

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); }
        ;

COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )                         { $setType(Token.SKIP); }
        ;

LPAREN  : '(';
RPAREN  : ')';
MULT    : '*';
PLUS    : '+';
```

```
MINUS   : '-';
DIV     : '/';
MOD     : '%';
COMMA : ',';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
ASGN    : '=';
PLUSEQ  : "+=";
MINUSEQ : "-=";
MULTEQ  : "*=";
DIVEQ   : "/=";
MODEQ   : "%=";
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
DPLUS   : "++";
DMINUS  : "--";
DBAR    : "||";
DAMP    : "&&";
NOT   : "!";

ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT)*
        ;

/* NUMBER example:
 * 1, 1e, 1., 1.e10, 1.1, 1.1e10
 */

NUMBER  : (DIGIT)+ ('.' (DIGIT)*)? (('E'|'e') ('+'|'-')? (DIGIT)+)? ;

STRING  : '"'!
              (  ~('"' | '\n')
               | ('"'!'"')
              )*
          '"'!
        ;

class AnimoAntlrParser extends Parser;

options {
k = 2;
buildAST = true;
exportVocab = AnimoAntlr;
}

tokens {
    STATEMENT;
    FUNC_CALL;
    EXPR_LIST;
    VAR_LIST;
    UPLUS;
    UMINUS;
```

```
    INCREMENT;
    DECREMENT;
    NEGATION;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

program : ( statement | func_def )* EOF!
        { #program = #([STATEMENT,"PROG"], program); }
;

statement
    : for_stmt
    | if_stmt
    | return_stmt
    | load_stmt
    | assignment
    | incr_decr_expr
    | func_call_stmt
    | LBRACE! (statement)* RBRACE!
        {#statement = #([STATEMENT,"STATEMENT"], statement); }
    ;

for_stmt : "for"^ LPAREN! ID ASGN! expression "to"! expression RPAREN! statement
;

if_stmt : "if"^ LPAREN! expression RPAREN! statement
        (options {greedy = true;}: "else"! statement )?
    ;

return_stmt : "return"^ (expression)? SEMI! ;

load_stmt : "include"^ STRING SEMI! ;

assignment
    : l_value ( ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^
        | DIVEQ^ | MODEQ^) expression SEMI!
    ;

func_call_stmt : func_call SEMI! ;

func_call
    : ID LPAREN! (expr_list)? RPAREN!
        { #func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
    ;

func_def
```

```
    : ( ("func"^ ID LPAREN! var_list RPAREN! func_body) => func_def_with_params
        | "func"^ ID LPAREN! RPAREN! func_body
        );

func_def_with_params
    : "func"^ ID LPAREN! var_list RPAREN! func_body
    ;

var_list
    : ID ( COMMA! ID )*
        { #var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
    | { #var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
    ;

func_body
    : LBRACE! (statement)* RBRACE!
        { #func_body = #([STATEMENT,"FUNC_BODY"], func_body); }
    ;

expression : logic_term ( DBAR^ logic_term )* ;

logic_term : relat_expr ( DAMP^ relat_expr )* ;

relat_expr : arith_expr ( (GE^ | LE^ | GT^
            | LT^ | EQ^ | NEQ^) arith_expr )? ;

arith_expr : arith_term ( (PLUS^ | MINUS^) arith_term )* ;

arith_term : arith_factor
        ( (MULT^ | DIV^ | MOD^) arith_factor )* ;

arith_factor
    : PLUS! r_value
        { #arith_factor = #([UPLUS,"UPLUS"], arith_factor); }
    | MINUS! r_value
        { #arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
    | DPLUS! l_value
        { #arith_factor = #([INCREMENT,"INCREMENT"], arith_factor); }
    | DMINUS! l_value
        { #arith_factor = #([DECREMENT,"DECREMENT"], arith_factor); }
    | NOT! r_value
        { #arith_factor = #([NEGATION,"NEGATION"], arith_factor); }
    | r_value
;

r_value : l_value | func_call | NUMBER | LPAREN! expression RPAREN! ;

l_value : ID ;

expr_list : expression (COMMA! expression)*
        { #expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
;

incr_decr_expr
    : DPLUS! l_value SEMI!
        { #incr_decr_expr = #([INCREMENT,"INCREMENT"], incr_decr_expr); }
    | DMINUS! l_value SEMI!
```

```
        { #incr_decr_expr = #([DECREMENT,"DECREMENT"], incr_decr_expr); }
;
```

```
/**********************************************
 * walker.g : the AST walker.
 * Authors:  Joshua Poritz, Natasha Shamis
 **********************************************/

{
import java.io.*;
import java.util.*;
}

class AnimoAntlrWalker extends TreeParser;
options{
    importVocab = AnimoAntlr;
}

{
    static AnimoDataType null_data = new AnimoDataType( "NULL" );
    AnimoCompiler cmpl = new AnimoCompiler();
}

expr returns [ AnimoDataType r ]
{
    AnimoDataType a, b, c;
    AnimoDataType[] x = null;
    String s = null;
    String[] sx = null;
    r = null_data;
}

      : #(DBAR a=expr b=expr)          { r = a.or(b); }
      | #(DAMP a=expr b=expr)          { r = a.and(b); }
        | #(NEGATION a=expr)          { r = a.not(); }
        | #(INCREMENT a=expr)         { r = a.incr(); }
        | #(DECREMENT a=expr)         { r = a.decr(); }
        | #(GE a=expr b=expr)         { r = a.ge( b ); }
        | #(LE a=expr b=expr)         { r = a.le( b ); }
        | #(GT a=expr b=expr)         { r = a.gt( b ); }
        | #(LT a=expr b=expr)         { r = a.lt( b ); }
        | #(EQ a=expr b=expr)         { r = a.eq( b ); }
        | #(NEQ a=expr b=expr)        { r = a.ne( b ); }
        | #(PLUS a=expr b=expr)       { r = a.plus( b ); }
        | #(MINUS a=expr b=expr)      { r = a.minus( b ); }
        | #(MULT a=expr b=expr)       { r = a.times( b ); }
        | #(DIV a=expr b=expr)        { r = a.divby( b ); }
        | #(MOD a=expr b=expr)        { r = a.modulus( b ); }
        | #(UPLUS a=expr)             { r = a; }
        | #(UMINUS a=expr)            { r = a.uminus(); }
        | #(PLUSEQ a=expr b=expr)     { r = a.add( b ); }
        | #(MINUSEQ a=expr b=expr)    { r = a.sub( b ); }
        | #(MULTEQ a=expr b=expr)     { r = a.mul( b ); }
        | #(DIVEQ a=expr b=expr)      { r = a.div( b ); }
        | #(MODEQ a=expr b=expr)      { r = a.rem( b ); }
        | #(ASGN a=expr b=expr)       { r = cmpl.assign( a, b ); }
        | #(FUNC_CALL a=fexpr (x=mexpr)?) {
            r = cmpl.funcInvoke( this, a, x );
        }
        | num:NUMBER                  { r = cmpl.getNumber( num.getText() ); }
```

```
        | id:ID                       { r = cmpl.getValue( id.getText() ); }
        | #("for" a=expr b=expr c=expr forbody:.)
        {
            if (b.typename().equals("unknown"))
                return b.error( "for: uninitialized variable in lower bound " );
            if (c.typename().equals("unknown"))
                return c.error( "for: uninitialized variable in upper bound " );

            if ( !( b instanceof AnimoFloatingPoint ) )
                return b.error( "for: lower bound is not an integer" );
            if ( !( c instanceof AnimoFloatingPoint ) )
                return c.error( "for: upper bound is not an integer" );

            double bVar = ((AnimoFloatingPoint)b).var;
            if (bVar != Math.floor(bVar))
                return b.error( "for: lower bound " + bVar + " is not an
integer" );

            double cVar = ((AnimoFloatingPoint)c).var;
            if (cVar != Math.floor(cVar))
                return c.error( "for: upper bound " + cVar + " is not an
integer" );

            cmpl.enterScope();
            cmpl.assign(a, b);
            AnimoDataType counter = cmpl.getValue( a.name );
            while (((AnimoFloatingPoint)counter).var <=
                ((AnimoFloatingPoint)c).var)
            {
                r = expr(#forbody);
                ((AnimoFloatingPoint)counter).var++;
            }
            ((AnimoFloatingPoint)counter).var--;
            cmpl.leaveScope();
        }
        | #("if" a=expr thenp:. (elsep:.)?)
            {
                if ( a.typename().equals("unknown") )
                    return a.error( "if: use of uninitialized value in " +
                                    "expression" );
                if ( !( a instanceof AnimoFloatingPoint ) )
                    return a.error( "if: expression is not a floating point" );
                cmpl.enterScope();
                if ( ((AnimoFloatingPoint)a).var != 0.0)
                    r = expr( #thenp );
                else if ( null != elsep )
                    r = expr( #elsep );
                cmpl.leaveScope();
            }
        | #(STATEMENT (stmt:. { if ( cmpl.canProceed() ) r = expr(#stmt); }
)*)
        | #("return" ( a=expr        { r = cmpl.rvalue( a ); }
                     )?
           )                        { cmpl.setReturn(); }
        | #("func" fname:ID sx=vlist fbody:.)
            { cmpl.funcRegister( fname.getText(), sx, #fbody ); }
        | #("include" s=string)      { cmpl.include( s ); }
```

```
        ;

mexpr returns [ AnimoDataType[] rv ]
{
    AnimoDataType a;
    rv = null;
    Vector v;
}
        : #(EXPR_LIST         { v = new Vector(); }
                ( a=expr      { v.add( a ); }
                )*
           )                  { rv = cmpl.convertExprList( v ); }
        | a=expr              { rv = new AnimoDataType[1]; rv[0] = a; }
        ;

vlist returns [ String[] sv ]
{
    Vector v;
    sv = null;
}
        : #(VAR_LIST     { v = new Vector(); }
             (s:ID       { v.add( s.getText() ); }
             )*
          )              { sv = cmpl.convertVarList( v ); }
        ;

fexpr returns [ AnimoDataType r ]
{
    r = null;
}
        : id:ID          { r = cmpl.getFunction(id.getText()); }
        ;

string returns [ String s ]
{
    s = null;
}
        : str:STRING     { s = str.getText(); }
        ;
```

```
/***********************************************
 * AnimoAntlrLexer.java: Created by Antlr
 ***********************************************/
// $ANTLR 2.7.5 (20050128): "grammar.g" -> "AnimoAntlrLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class AnimoAntlrLexer extends antlr.CharScanner implements
AnimoAntlrTokenTypes, TokenStream
 {

    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
public AnimoAntlrLexer(InputStream in) {
      this(new ByteBuffer(in));
}
public AnimoAntlrLexer(Reader in) {
      this(new CharBuffer(in));
}
public AnimoAntlrLexer(InputBuffer ib) {
      this(new LexerSharedInputState(ib));
}
public AnimoAntlrLexer(LexerSharedInputState state) {
      super(state);
      caseSensitiveLiterals = true;
      setCaseSensitive(true);
      literals = new Hashtable();
      literals.put(new ANTLRHashString("if", this), new Integer(51));
```

```
      literals.put(new ANTLRHashString("for", this), new Integer(49));
      literals.put(new ANTLRHashString("func", this), new Integer(55));
      literals.put(new ANTLRHashString("else", this), new Integer(52));
      literals.put(new ANTLRHashString("include", this), new Integer(54));
      literals.put(new ANTLRHashString("return", this), new Integer(53));
      literals.put(new ANTLRHashString("to", this), new Integer(50));
}

public Token nextToken() throws TokenStreamException {
      Token theRetToken=null;
tryAgain:
      for (;;) {
            Token _token = null;
            int _ttype = Token.INVALID_TYPE;
            resetText();
            try {   // for char stream error handling
                  try {   // for lexical error handling
                        switch ( LA(1)) {
                        case '\t':  case ' ':
                        {
                              mWS(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '\n':  case '\r':
                        {
                              mNL(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '(':
                        {
                              mLPAREN(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case ')':
                        {
                              mRPAREN(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case ',':
                        {
                              mCOMMA(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case ';':
                        {
                              mSEMI(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '{':
                        {
                              mLBRACE(true);
```

```
      theRetToken=_returnToken;
      break;
}
case '}':
{
      mRBRACE(true);
      theRetToken=_returnToken;
      break;
}
case '|':
{
      mDBAR(true);
      theRetToken=_returnToken;
      break;
}
case '&':
{
      mDAMP(true);
      theRetToken=_returnToken;
      break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case '_':  case 'a':
case 'b':  case 'c':  case 'd':  case 'e':
case 'f':  case 'g':  case 'h':  case 'i':
case 'j':  case 'k':  case 'l':  case 'm':
case 'n':  case 'o':  case 'p':  case 'q':
case 'r':  case 's':  case 't':  case 'u':
case 'v':  case 'w':  case 'x':  case 'y':
case 'z':
{
      mID(true);
      theRetToken=_returnToken;
      break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
      mNUMBER(true);
      theRetToken=_returnToken;
      break;
}
case '"':
{
      mSTRING(true);
      theRetToken=_returnToken;
      break;
}
default:
      if ((LA(1)=='/') && (LA(2)=='*'||LA(2)=='/')) {
            mCOMMENT(true);
```

```
      theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='=')) {
      mPLUSEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='=')) {
      mMINUSEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (LA(2)=='=')) {
      mMULTEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (LA(2)=='=')) {
      mDIVEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='%') && (LA(2)=='=')) {
      mMODEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (LA(2)=='=')) {
      mGE(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (LA(2)=='=')) {
      mLE(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='=') && (LA(2)=='=')) {
      mEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='!') && (LA(2)=='=')) {
      mNEQ(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='+')) {
      mDPLUS(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='-')) {
      mDMINUS(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (true)) {
      mMULT(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (true)) {
      mPLUS(true);
      theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (true)) {
      mMINUS(true);
      theRetToken=_returnToken;
```

```
                              }
                              else if ((LA(1)=='/') && (true)) {
                                    mDIV(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='%') && (true)) {
                                    mMOD(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='=') && (true)) {
                                    mASGN(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='>') && (true)) {
                                    mGT(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='<') && (true)) {
                                    mLT(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='!') && (true)) {
                                    mNOT(true);
                                    theRetToken=_returnToken;
                              }
                        else {
                              if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken =
makeToken(Token.EOF_TYPE);}
                        else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
                        }
                        }
                        if ( _returnToken==null ) continue tryAgain; // found
SKIP token
                        _ttype = _returnToken.getType();
                        _returnToken.setType(_ttype);
                        return _returnToken;
                  }
                  catch (RecognitionException e) {
                        throw new TokenStreamRecognitionException(e);
                  }
            }
            catch (CharStreamException cse) {
                  if ( cse instanceof CharStreamIOException ) {
                        throw new
TokenStreamIOException(((CharStreamIOException)cse).io);
                  }
                  else {
                        throw new TokenStreamException(cse.getMessage());
                  }
            }
      }
}

      protected final void mALPHA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
```

```
            _ttype = ALPHA;
            int _saveIndex;

            switch ( LA(1)) {
            case 'a':  case 'b':  case 'c':  case 'd':
            case 'e':  case 'f':  case 'g':  case 'h':
            case 'i':  case 'j':  case 'k':  case 'l':
            case 'm':  case 'n':  case 'o':  case 'p':
            case 'q':  case 'r':  case 's':  case 't':
            case 'u':  case 'v':  case 'w':  case 'x':
            case 'y':  case 'z':
            {
                  matchRange('a','z');
                  break;
            }
            case 'A':  case 'B':  case 'C':  case 'D':
            case 'E':  case 'F':  case 'G':  case 'H':
            case 'I':  case 'J':  case 'K':  case 'L':
            case 'M':  case 'N':  case 'O':  case 'P':
            case 'Q':  case 'R':  case 'S':  case 'T':
            case 'U':  case 'V':  case 'W':  case 'X':
            case 'Y':  case 'Z':
            {
                  matchRange('A','Z');
                  break;
            }
            case '_':
            {
                  match('_');
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      protected final void mDIGIT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIGIT;
            int _saveIndex;

            matchRange('0','9');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
```

```
            _returnToken = _token;
      }

      public final void mWS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = WS;
            int _saveIndex;

            {
            int _cnt5=0;
            _loop5:
            do {
                  switch ( LA(1)) {
                  case ' ':
                  {
                        match(' ');
                        break;
                  }
                  case '\t':
                  {
                        match('\t');
                        break;
                  }
                  default:
                  {
                        if ( _cnt5>=1 ) { break _loop5; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                  }
                  }
                  _cnt5++;
            } while (true);
            }
            if ( inputState.guessing==0 ) {
                  _ttype = Token.SKIP;
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NL;
            int _saveIndex;

            {
            boolean synPredMatched9 = false;
            if (((LA(1)=='\r') && (LA(2)=='\n'))) {
                  int _m9 = mark();
                  synPredMatched9 = true;
                  inputState.guessing++;
```

```
                  try {
                        {
                        match('\r');
                        match('\n');
                        }
                  }
                  catch (RecognitionException pe) {
                        synPredMatched9 = false;
                  }
                  rewind(_m9);
                  inputState.guessing--;
            }
            if ( synPredMatched9 ) {
                  match('\r');
                  match('\n');
            }
            else if ((LA(1)=='\n')) {
                  match('\n');
            }
            else if ((LA(1)=='\r') && (true)) {
                  match('\r');
            }
            else {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }

            }
            if ( inputState.guessing==0 ) {
                  _ttype = Token.SKIP; newline();
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMENT;
            int _saveIndex;

            {
            if ((LA(1)=='/') && (LA(2)=='*')) {
                  match("/*");
                  {
                  _loop15:
                  do {
                        // nongreedy exit test
                        if ((LA(1)=='*') && (LA(2)=='/')) break _loop15;
                        if ((_tokenSet_0.member(LA(1))) && ((LA(2) >= '\u0003'
&& LA(2) <= '\u00ff'))) {
                              {
                              match(_tokenSet_0);
```

```
                        }
                    }
                    else if ((LA(1)=='\n'||LA(1)=='\r')) {
                        {
                        mNL(false);
                        }
                    }
                    else {
                        break _loop15;
                    }

                } while (true);
                }
                match("*/");
            }
            else if ((LA(1)=='/') && (LA(2)=='/')) {
                match("//");
                {
                _loop18:
                do {
                    if ((_tokenSet_0.member(LA(1)))) {
                        {
                        match(_tokenSet_0);
                        }
                    }
                    else {
                        break _loop18;
                    }

                } while (true);
                }
                {
                mNL(false);
                }
            }
            else {
                throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }

            }
            if ( inputState.guessing==0 ) {
                _ttype = Token.SKIP;
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mLPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LPAREN;
            int _saveIndex;
```

```
            match('(');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RPAREN;
            int _saveIndex;

            match(')');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMULT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MULT;
            int _saveIndex;

            match('*');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mPLUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUS;
            int _saveIndex;

            match('+');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMINUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
```

```
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MINUS;
            int _saveIndex;

            match('-');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDIV(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIV;
            int _saveIndex;

            match('/');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMOD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MOD;
            int _saveIndex;

            match('%');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMA;
            int _saveIndex;

            match(',');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
	public final void mSEMI(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SEMI;
		int _saveIndex;

		match(';');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LBRACE;
		int _saveIndex;

		match('{');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RBRACE;
		int _saveIndex;

		match('}');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mASGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ASGN;
		int _saveIndex;

		match('=');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
```

```
            }
            _returnToken = _token;
      }

      public final void mPLUSEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUSEQ;
            int _saveIndex;

            match("+=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMINUSEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MINUSEQ;
            int _saveIndex;

            match("-=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMULTEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MULTEQ;
            int _saveIndex;

            match("*=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDIVEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIVEQ;
            int _saveIndex;

            match("/=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMODEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MODEQ;
            int _saveIndex;

            match("%=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mGE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = GE;
            int _saveIndex;

            match(">=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LE;
            int _saveIndex;

            match("<=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mGT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = GT;
            int _saveIndex;
```

```
            match('>');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LT;
            int _saveIndex;

            match('<');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = EQ;
            int _saveIndex;

            match("==");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NEQ;
            int _saveIndex;

            match("!=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDPLUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
```

```
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DPLUS;
            int _saveIndex;

            match("++");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDMINUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DMINUS;
            int _saveIndex;

            match("--");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDBAR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DBAR;
            int _saveIndex;

            match("||");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDAMP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DAMP;
            int _saveIndex;

            match("&&");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
      public final void mNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NOT;
            int _saveIndex;

            match("!");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mID(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = ID;
            int _saveIndex;

            mALPHA(false);
            {
            _loop50:
            do {
                  switch ( LA(1)) {
                  case 'A':  case 'B':  case 'C':  case 'D':
                  case 'E':  case 'F':  case 'G':  case 'H':
                  case 'I':  case 'J':  case 'K':  case 'L':
                  case 'M':  case 'N':  case 'O':  case 'P':
                  case 'Q':  case 'R':  case 'S':  case 'T':
                  case 'U':  case 'V':  case 'W':  case 'X':
                  case 'Y':  case 'Z':  case '_':  case 'a':
                  case 'b':  case 'c':  case 'd':  case 'e':
                  case 'f':  case 'g':  case 'h':  case 'i':
                  case 'j':  case 'k':  case 'l':  case 'm':
                  case 'n':  case 'o':  case 'p':  case 'q':
                  case 'r':  case 's':  case 't':  case 'u':
                  case 'v':  case 'w':  case 'x':  case 'y':
                  case 'z':
                  {
                        mALPHA(false);
                        break;
                  }
                  case '0':  case '1':  case '2':  case '3':
                  case '4':  case '5':  case '6':  case '7':
                  case '8':  case '9':
                  {
                        mDIGIT(false);
                        break;
                  }
                  default:
                  {
                        break _loop50;
                  }
                  }
```

```
            } while (true);
            }
            _ttype = testLiteralsTable(_ttype);
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNUMBER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NUMBER;
            int _saveIndex;

            {
            int _cnt53=0;
            _loop53:
            do {
                  if (((LA(1) >= '0' && LA(1) <= '9'))) {
                        mDIGIT(false);
                  }
                  else {
                        if ( _cnt53>=1 ) { break _loop53; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                  }

                  _cnt53++;
            } while (true);
            }
            {
            if ((LA(1)=='.')) {
                  match('.');
                  {
                  _loop56:
                  do {
                        if (((LA(1) >= '0' && LA(1) <= '9'))) {
                              mDIGIT(false);
                        }
                        else {
                              break _loop56;
                        }

                  } while (true);
                  }
            }
            else {
            }

            }
            {
            if ((LA(1)=='E'||LA(1)=='e')) {
                  {
                  switch ( LA(1)) {
```

```
                    case 'E':
                    {
                          match('E');
                          break;
                    }
                    case 'e':
                    {
                          match('e');
                          break;
                    }
                    default:
                    {
                          throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                    }
                    }
                    }
                    {
                    switch ( LA(1)) {
                    case '+':
                    {
                          match('+');
                          break;
                    }
                    case '-':
                    {
                          match('-');
                          break;
                    }
                    case '0':  case '1':  case '2':  case '3':
                    case '4':  case '5':  case '6':  case '7':
                    case '8':  case '9':
                    {
                          break;
                    }
                    default:
                    {
                          throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                    }
                    }
                    }
                    {
                    int _cnt61=0;
                    _loop61:
                    do {
                          if (((LA(1) >= '0' && LA(1) <= '9'))) {
                                mDIGIT(false);
                          }
                          else {
                                if ( _cnt61>=1 ) { break _loop61; } else {throw
new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                          }

                          _cnt61++;
                    } while (true);
```

```
                  }
            }
            else {
            }

            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mSTRING(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = STRING;
            int _saveIndex;

            _saveIndex=text.length();
            match('"');
            text.setLength(_saveIndex);
            {
            _loop66:
            do {
                  if ((LA(1)=='"') && (LA(2)=='")) {
                        {
                        _saveIndex=text.length();
                        match('"');
                        text.setLength(_saveIndex);
                        match('"');
                        }
                  }
                  else if ((_tokenSet_1.member(LA(1)))) {
                        {
                        match(_tokenSet_1);
                        }
                  }
                  else {
                        break _loop66;
                  }

            } while (true);
            }
            _saveIndex=text.length();
            match('"');
            text.setLength(_saveIndex);
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
private static final long[] mk_tokenSet_0() {
      long[] data = new long[8];
      data[0]=-9224L;
      for (int i = 1; i<=3; i++) { data[i]=-1L; }
      return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
      long[] data = new long[8];
      data[0]=-17179870216L;
      for (int i = 1; i<=3; i++) { data[i]=-1L; }
      return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());

}
```

```
/***********************************************************
 * AnimoAntlrParser.java: Created by Antlr
 ***********************************************************/

// $ANTLR 2.7.5 (20050128): "grammar.g" -> "AnimoAntlrParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class AnimoAntlrParser extends antlr.LLkParser       implements
AnimoAntlrTokenTypes
 {

    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }

protected AnimoAntlrParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public AnimoAntlrParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected AnimoAntlrParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
```

```
public AnimoAntlrParser(TokenStream lexer) {
  this(lexer,2);
}

public AnimoAntlrParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

      public final void program() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST program_AST = null;

            try {      // for error handling
                  {
                  _loop69:
                  do {
                        switch ( LA(1)) {
                        case LBRACE:
                        case DPLUS:
                        case DMINUS:
                        case ID:
                        case LITERAL_for:
                        case LITERAL_if:
                        case LITERAL_return:
                        case LITERAL_include:
                        {
                              statement();
                              astFactory.addASTChild(currentAST, returnAST);
                              break;
                        }
                        case LITERAL_func:
                        {
                              func_def();
                              astFactory.addASTChild(currentAST, returnAST);
                              break;
                        }
                        default:
                        {
                              break _loop69;
                        }
                        }
                  } while (true);
                  }
                  match(Token.EOF_TYPE);
                  if ( inputState.guessing==0 ) {
                        program_AST = (AST)currentAST.root;
                        program_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(STATEMENT,"PROG")).add(program_AST));
                        currentAST.root = program_AST;
                        currentAST.child = program_AST!=null
&&program_AST.getFirstChild()!=null ?
```

```
                        program_AST.getFirstChild() : program_AST;
                  currentAST.advanceChildToEnd();
            }
            program_AST = (AST)currentAST.root;
      }
      catch (RecognitionException ex) {
            if (inputState.guessing==0) {
                  reportError(ex);
                  recover(ex,_tokenSet_0);
            } else {
              throw ex;
            }
      }
      returnAST = program_AST;
}

      public final void statement() throws RecognitionException,
TokenStreamException {

      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST statement_AST = null;

      try {      // for error handling
            switch ( LA(1)) {
            case LITERAL_for:
            {
                  for_stmt();
                  astFactory.addASTChild(currentAST, returnAST);
                  statement_AST = (AST)currentAST.root;
                  break;
            }
            case LITERAL_if:
            {
                  if_stmt();
                  astFactory.addASTChild(currentAST, returnAST);
                  statement_AST = (AST)currentAST.root;
                  break;
            }
            case LITERAL_return:
            {
                  return_stmt();
                  astFactory.addASTChild(currentAST, returnAST);
                  statement_AST = (AST)currentAST.root;
                  break;
            }
            case LITERAL_include:
            {
                  load_stmt();
                  astFactory.addASTChild(currentAST, returnAST);
                  statement_AST = (AST)currentAST.root;
                  break;
            }
            case DPLUS:
            case DMINUS:
            {
                  incr_decr_expr();
```

```
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                  }
                  case LBRACE:
                  {
                        match(LBRACE);
                        {
                        _loop72:
                        do {
                              if ((_tokenSet_1.member(LA(1)))) {
                                    statement();
                                    astFactory.addASTChild(currentAST,
returnAST);
                              }
                              else {
                                    break _loop72;
                              }

                        } while (true);
                        }
                        match(RBRACE);
                        if ( inputState.guessing==0 ) {
                              statement_AST = (AST)currentAST.root;
                              statement_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(STATEMENT,"STATEMENT")).add(statement_AST));
                              currentAST.root = statement_AST;
                              currentAST.child = statement_AST!=null
&&statement_AST.getFirstChild()!=null ?
                                    statement_AST.getFirstChild() :
statement_AST;
                              currentAST.advanceChildToEnd();
                        }
                        statement_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                        if ((LA(1)==ID) && ((LA(2) >= ASGN && LA(2) <= MODEQ)))
{
                              assignment();
                              astFactory.addASTChild(currentAST, returnAST);
                              statement_AST = (AST)currentAST.root;
                        }
                        else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
                              func_call_stmt();
                              astFactory.addASTChild(currentAST, returnAST);
                              statement_AST = (AST)currentAST.root;
                        }
                  else {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_2);
```

```
                } else {
                  throw ex;
                }
          }
          returnAST = statement_AST;
    }

    public final void func_def() throws RecognitionException,
TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST func_def_AST = null;

          try {      // for error handling
                {
                boolean synPredMatched87 = false;
                if (((LA(1)==LITERAL_func) && (LA(2)==ID))) {
                      int _m87 = mark();
                      synPredMatched87 = true;
                      inputState.guessing++;
                      try {
                            {
                            match(LITERAL_func);
                            match(ID);
                            match(LPAREN);
                            var_list();
                            match(RPAREN);
                            func_body();
                            }
                      }
                      catch (RecognitionException pe) {
                            synPredMatched87 = false;
                      }
                      rewind(_m87);
                      inputState.guessing--;
                }
                if ( synPredMatched87 ) {
                      func_def_with_params();
                      astFactory.addASTChild(currentAST, returnAST);
                }
                else if ((LA(1)==LITERAL_func) && (LA(2)==ID)) {
                      AST tmp4_AST = null;
                      tmp4_AST = astFactory.create(LT(1));
                      astFactory.makeASTRoot(currentAST, tmp4_AST);
                      match(LITERAL_func);
                      AST tmp5_AST = null;
                      tmp5_AST = astFactory.create(LT(1));
                      astFactory.addASTChild(currentAST, tmp5_AST);
                      match(ID);
                      match(LPAREN);
                      match(RPAREN);
                      func_body();
                      astFactory.addASTChild(currentAST, returnAST);
                }
                else {
                      throw new NoViableAltException(LT(1), getFilename());
```

```
                }

                }
                func_def_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                    reportError(ex);
                    recover(ex,_tokenSet_3);
                } else {
                  throw ex;
                }
            }
            returnAST = func_def_AST;
        }

        public final void for_stmt() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST for_stmt_AST = null;

            try {      // for error handling
                AST tmp8_AST = null;
                tmp8_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp8_AST);
                match(LITERAL_for);
                match(LPAREN);
                AST tmp10_AST = null;
                tmp10_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp10_AST);
                match(ID);
                match(ASGN);
                expression();
                astFactory.addASTChild(currentAST, returnAST);
                match(LITERAL_to);
                expression();
                astFactory.addASTChild(currentAST, returnAST);
                match(RPAREN);
                statement();
                astFactory.addASTChild(currentAST, returnAST);
                for_stmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                    reportError(ex);
                    recover(ex,_tokenSet_2);
                } else {
                  throw ex;
                }
            }
            returnAST = for_stmt_AST;
        }

        public final void if_stmt() throws RecognitionException,
TokenStreamException {
```

```
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST if_stmt_AST = null;

            try {      // for error handling
                  AST tmp14_AST = null;
                  tmp14_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp14_AST);
                  match(LITERAL_if);
                  match(LPAREN);
                  expression();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RPAREN);
                  statement();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  if ((LA(1)==LITERAL_else) && (_tokenSet_1.member(LA(2)))) {
                        match(LITERAL_else);
                        statement();
                        astFactory.addASTChild(currentAST, returnAST);
                  }
                  else if ((_tokenSet_2.member(LA(1))) &&
(_tokenSet_4.member(LA(2)))) {
                  }
                  else {
                        throw new NoViableAltException(LT(1), getFilename());
                  }

                  }
                  if_stmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_2);
                  } else {
                    throw ex;
                  }
            }
            returnAST = if_stmt_AST;
      }

      public final void return_stmt() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST return_stmt_AST = null;

            try {      // for error handling
                  AST tmp18_AST = null;
                  tmp18_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp18_AST);
                  match(LITERAL_return);
                  {
                  switch ( LA(1)) {
```

```
                case LPAREN:
                case PLUS:
                case MINUS:
                case DPLUS:
                case DMINUS:
                case NOT:
                case ID:
                case NUMBER:
                {
                      expression();
                      astFactory.addASTChild(currentAST, returnAST);
                      break;
                }
                case SEMI:
                {
                      break;
                }
                default:
                {
                      throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                match(SEMI);
                return_stmt_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                      reportError(ex);
                      recover(ex,_tokenSet_2);
                } else {
                  throw ex;
                }
          }
          returnAST = return_stmt_AST;
    }

    public final void load_stmt() throws RecognitionException,
TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST load_stmt_AST = null;

          try {      // for error handling
                AST tmp20_AST = null;
                tmp20_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp20_AST);
                match(LITERAL_include);
                AST tmp21_AST = null;
                tmp21_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp21_AST);
                match(STRING);
                match(SEMI);
                load_stmt_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
```

```
                if (inputState.guessing==0) {
                      reportError(ex);
                      recover(ex,_tokenSet_2);
                } else {
                  throw ex;
                }
          }
          returnAST = load_stmt_AST;
    }

    public final void assignment() throws RecognitionException,
TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST assignment_AST = null;

          try {      // for error handling
                l_value();
                astFactory.addASTChild(currentAST, returnAST);
                {
                switch ( LA(1)) {
                case ASGN:
                {
                      AST tmp23_AST = null;
                      tmp23_AST = astFactory.create(LT(1));
                      astFactory.makeASTRoot(currentAST, tmp23_AST);
                      match(ASGN);
                      break;
                }
                case PLUSEQ:
                {
                      AST tmp24_AST = null;
                      tmp24_AST = astFactory.create(LT(1));
                      astFactory.makeASTRoot(currentAST, tmp24_AST);
                      match(PLUSEQ);
                      break;
                }
                case MINUSEQ:
                {
                      AST tmp25_AST = null;
                      tmp25_AST = astFactory.create(LT(1));
                      astFactory.makeASTRoot(currentAST, tmp25_AST);
                      match(MINUSEQ);
                      break;
                }
                case MULTEQ:
                {
                      AST tmp26_AST = null;
                      tmp26_AST = astFactory.create(LT(1));
                      astFactory.makeASTRoot(currentAST, tmp26_AST);
                      match(MULTEQ);
                      break;
                }
                case DIVEQ:
                {
                      AST tmp27_AST = null;
```

```
                        tmp27_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp27_AST);
                        match(DIVEQ);
                        break;
                  }
                  case MODEQ:
                  {
                        AST tmp28_AST = null;
                        tmp28_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp28_AST);
                        match(MODEQ);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
                  }
                  expression();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(SEMI);
                  assignment_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_2);
                  } else {
                    throw ex;
                  }
            }
            returnAST = assignment_AST;
      }

      public final void incr_decr_expr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST incr_decr_expr_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case DPLUS:
                  {
                        match(DPLUS);
                        l_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(SEMI);
                        if ( inputState.guessing==0 ) {
                              incr_decr_expr_AST = (AST)currentAST.root;
                              incr_decr_expr_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(INCREMENT,"INCREMENT")).add(incr_decr_expr_AS
T));
                              currentAST.root = incr_decr_expr_AST;
```

```
                              currentAST.child = incr_decr_expr_AST!=null
&&incr_decr_expr_AST.getFirstChild()!=null ?
                                    incr_decr_expr_AST.getFirstChild() :
incr_decr_expr_AST;
                              currentAST.advanceChildToEnd();
                        }
                        incr_decr_expr_AST = (AST)currentAST.root;
                        break;
                  }
                  case DMINUS:
                  {
                        match(DMINUS);
                        l_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(SEMI);
                        if ( inputState.guessing==0 ) {
                              incr_decr_expr_AST = (AST)currentAST.root;
                              incr_decr_expr_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(DECREMENT,"DECREMENT")).add(incr_decr_expr_AS
T));
                              currentAST.root = incr_decr_expr_AST;
                              currentAST.child = incr_decr_expr_AST!=null
&&incr_decr_expr_AST.getFirstChild()!=null ?
                                    incr_decr_expr_AST.getFirstChild() :
incr_decr_expr_AST;
                              currentAST.advanceChildToEnd();
                        }
                        incr_decr_expr_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_2);
                  } else {
                    throw ex;
                  }
            }
            returnAST = incr_decr_expr_AST;
      }

      public final void func_call_stmt() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST func_call_stmt_AST = null;

            try {      // for error handling
                  func_call();
                  astFactory.addASTChild(currentAST, returnAST);
```

```
                match(SEMI);
                func_call_stmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                    reportError(ex);
                    recover(ex,_tokenSet_2);
                } else {
                  throw ex;
                }
            }
            returnAST = func_call_stmt_AST;
        }

        public final void expression() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expression_AST = null;

            try {      // for error handling
                logic_term();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop97:
                do {
                    if ((LA(1)==DBAR)) {
                        AST tmp35_AST = null;
                        tmp35_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp35_AST);
                        match(DBAR);
                        logic_term();
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                    else {
                        break _loop97;
                    }

                } while (true);
                }
                expression_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                    reportError(ex);
                    recover(ex,_tokenSet_5);
                } else {
                  throw ex;
                }
            }
            returnAST = expression_AST;
        }

        public final void l_value() throws RecognitionException,
TokenStreamException {
```

```
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST l_value_AST = null;

            try {      // for error handling
                  AST tmp36_AST = null;
                  tmp36_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp36_AST);
                  match(ID);
                  l_value_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_6);
                  } else {
                    throw ex;
                  }
            }
            returnAST = l_value_AST;
      }

      public final void func_call() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST func_call_AST = null;

            try {      // for error handling
                  AST tmp37_AST = null;
                  tmp37_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp37_AST);
                  match(ID);
                  match(LPAREN);
                  {
                  switch ( LA(1)) {
                  case LPAREN:
                  case PLUS:
                  case MINUS:
                  case DPLUS:
                  case DMINUS:
                  case NOT:
                  case ID:
                  case NUMBER:
                  {
                        expr_list();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case RPAREN:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
```

```
                  }
                  }
                  }
                  match(RPAREN);
                  if ( inputState.guessing==0 ) {
                        func_call_AST = (AST)currentAST.root;
                        func_call_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(FUNC_CALL,"FUNC_CALL")).add(func_call_AST));
                        currentAST.root = func_call_AST;
                        currentAST.child = func_call_AST!=null
&&func_call_AST.getFirstChild()!=null ?
                              func_call_AST.getFirstChild() : func_call_AST;
                        currentAST.advanceChildToEnd();
                  }
                  func_call_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_7);
                  } else {
                    throw ex;
                  }
            }
            returnAST = func_call_AST;
      }

      public final void expr_list() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expr_list_AST = null;

            try {      // for error handling
                  expression();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop117:
                  do {
                        if ((LA(1)==COMMA)) {
                              match(COMMA);
                              expression();
                              astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                              break _loop117;
                        }

                  } while (true);
                  }
                  if ( inputState.guessing==0 ) {
                        expr_list_AST = (AST)currentAST.root;
                        expr_list_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(EXPR_LIST,"EXPR_LIST")).add(expr_list_AST));
                        currentAST.root = expr_list_AST;
```

```
                        currentAST.child = expr_list_AST!=null
&&expr_list_AST.getFirstChild()!=null ?
                              expr_list_AST.getFirstChild() : expr_list_AST;
                        currentAST.advanceChildToEnd();
                  }
                  expr_list_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_8);
                  } else {
                    throw ex;
                  }
            }
            returnAST = expr_list_AST;
      }

      public final void var_list() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST var_list_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case ID:
                  {
                        AST tmp41_AST = null;
                        tmp41_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp41_AST);
                        match(ID);
                        {
                        _loop91:
                        do {
                              if ((LA(1)==COMMA)) {
                                    match(COMMA);
                                    AST tmp43_AST = null;
                                    tmp43_AST = astFactory.create(LT(1));
                                    astFactory.addASTChild(currentAST,
tmp43_AST);
                                    match(ID);
                              }
                              else {
                                    break _loop91;
                              }

                        } while (true);
                        }
                        if ( inputState.guessing==0 ) {
                              var_list_AST = (AST)currentAST.root;
                              var_list_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(VAR_LIST,"VAR_LIST")).add(var_list_AST));
                              currentAST.root = var_list_AST;
                              currentAST.child = var_list_AST!=null
&&var_list_AST.getFirstChild()!=null ?
```

```
                                    var_list_AST.getFirstChild() : var_list_AST;
                              currentAST.advanceChildToEnd();
                        }
                        var_list_AST = (AST)currentAST.root;
                        break;
                  }
                  case RPAREN:
                  {
                        if ( inputState.guessing==0 ) {
                              var_list_AST = (AST)currentAST.root;
                              var_list_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(VAR_LIST,"VAR_LIST")).add(var_list_AST));
                              currentAST.root = var_list_AST;
                              currentAST.child = var_list_AST!=null
&&var_list_AST.getFirstChild()!=null ?
                                    var_list_AST.getFirstChild() : var_list_AST;
                              currentAST.advanceChildToEnd();
                        }
                        var_list_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_8);
                  } else {
                    throw ex;
                  }
            }
            returnAST = var_list_AST;
      }

      public final void func_body() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST func_body_AST = null;

            try {      // for error handling
                  match(LBRACE);
                  {
                  _loop94:
                  do {
                        if ((_tokenSet_1.member(LA(1)))) {
                              statement();
                              astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                              break _loop94;
                        }
```

```
                  } while (true);
                  }
                  match(RBRACE);
                  if ( inputState.guessing==0 ) {
                        func_body_AST = (AST)currentAST.root;
                        func_body_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(STATEMENT,"FUNC_BODY")).add(func_body_AST));
                        currentAST.root = func_body_AST;
                        currentAST.child = func_body_AST!=null
&&func_body_AST.getFirstChild()!=null ?
                              func_body_AST.getFirstChild() : func_body_AST;
                        currentAST.advanceChildToEnd();
                  }
                  func_body_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_3);
                  } else {
                    throw ex;
                  }
            }
            returnAST = func_body_AST;
      }

      public final void func_def_with_params() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST func_def_with_params_AST = null;

            try {      // for error handling
                  AST tmp46_AST = null;
                  tmp46_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp46_AST);
                  match(LITERAL_func);
                  AST tmp47_AST = null;
                  tmp47_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp47_AST);
                  match(ID);
                  match(LPAREN);
                  var_list();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RPAREN);
                  func_body();
                  astFactory.addASTChild(currentAST, returnAST);
                  func_def_with_params_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_3);
                  } else {
                    throw ex;
```

```
                }
            }
            returnAST = func_def_with_params_AST;
      }

      public final void logic_term() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST logic_term_AST = null;

            try {      // for error handling
                  relat_expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop100:
                  do {
                        if ((LA(1)==DAMP)) {
                              AST tmp50_AST = null;
                              tmp50_AST = astFactory.create(LT(1));
                              astFactory.makeASTRoot(currentAST, tmp50_AST);
                              match(DAMP);
                              relat_expr();
                              astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                              break _loop100;
                        }

                  } while (true);
                  }
                  logic_term_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_9);
                  } else {
                    throw ex;
                  }
            }
            returnAST = logic_term_AST;
      }

      public final void relat_expr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST relat_expr_AST = null;

            try {      // for error handling
                  arith_expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  switch ( LA(1)) {
```

```
case GE:
case LE:
case GT:
case LT:
case EQ:
case NEQ:
{
	{
	switch ( LA(1)) {
	case GE:
	{
		AST tmp51_AST = null;
		tmp51_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp51_AST);
		match(GE);
		break;
	}
	case LE:
	{
		AST tmp52_AST = null;
		tmp52_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp52_AST);
		match(LE);
		break;
	}
	case GT:
	{
		AST tmp53_AST = null;
		tmp53_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp53_AST);
		match(GT);
		break;
	}
	case LT:
	{
		AST tmp54_AST = null;
		tmp54_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp54_AST);
		match(LT);
		break;
	}
	case EQ:
	{
		AST tmp55_AST = null;
		tmp55_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp55_AST);
		match(EQ);
		break;
	}
	case NEQ:
	{
		AST tmp56_AST = null;
		tmp56_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp56_AST);
		match(NEQ);
		break;
	}
```

```
                        default:
                        {
                              throw new NoViableAltException(LT(1),
getFilename());
                        }
                        }
                        }
                        arith_expr();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case RPAREN:
                  case COMMA:
                  case SEMI:
                  case DBAR:
                  case DAMP:
                  case LITERAL_to:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
                  }
                  relat_expr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_10);
                  } else {
                    throw ex;
                  }
            }
            returnAST = relat_expr_AST;
      }

      public final void arith_expr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST arith_expr_AST = null;

            try {      // for error handling
                  arith_term();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop107:
                  do {
                        if ((LA(1)==PLUS||LA(1)==MINUS)) {
                              {
                              switch ( LA(1)) {
                              case PLUS:
                              {
```

```
                                    AST tmp57_AST = null;
                                    tmp57_AST = astFactory.create(LT(1));
                                    astFactory.makeASTRoot(currentAST,
tmp57_AST);
                                    match(PLUS);
                                    break;
                              }
                              case MINUS:
                              {
                                    AST tmp58_AST = null;
                                    tmp58_AST = astFactory.create(LT(1));
                                    astFactory.makeASTRoot(currentAST,
tmp58_AST);
                                    match(MINUS);
                                    break;
                              }
                              default:
                              {
                                    throw new NoViableAltException(LT(1),
getFilename());
                              }
                              }
                              }
                              arith_term();
                              astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                              break _loop107;
                        }

                  } while (true);
                  }
                  arith_expr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_11);
                  } else {
                    throw ex;
                  }
            }
            returnAST = arith_expr_AST;
      }

      public final void arith_term() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST arith_term_AST = null;

            try {      // for error handling
                  arith_factor();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop111:
```

```
                do {
                      if ((LA(1)==MULT||LA(1)==DIV||LA(1)==MOD)) {
                            {
                            switch ( LA(1)) {
                            case MULT:
                            {
                                  AST tmp59_AST = null;
                                  tmp59_AST = astFactory.create(LT(1));
                                  astFactory.makeASTRoot(currentAST,
tmp59_AST);
                                  match(MULT);
                                  break;
                            }
                            case DIV:
                            {
                                  AST tmp60_AST = null;
                                  tmp60_AST = astFactory.create(LT(1));
                                  astFactory.makeASTRoot(currentAST,
tmp60_AST);
                                  match(DIV);
                                  break;
                            }
                            case MOD:
                            {
                                  AST tmp61_AST = null;
                                  tmp61_AST = astFactory.create(LT(1));
                                  astFactory.makeASTRoot(currentAST,
tmp61_AST);
                                  match(MOD);
                                  break;
                            }
                            default:
                            {
                                  throw new NoViableAltException(LT(1),
getFilename());
                            }
                            }
                            }
                            arith_factor();
                            astFactory.addASTChild(currentAST, returnAST);
                      }
                      else {
                            break _loop111;
                      }

                } while (true);
                }
                arith_term_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
                if (inputState.guessing==0) {
                      reportError(ex);
                      recover(ex,_tokenSet_12);
                } else {
                  throw ex;
                }
          }
```

```
            returnAST = arith_term_AST;
      }

      public final void arith_factor() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST arith_factor_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case PLUS:
                  {
                        match(PLUS);
                        r_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        if ( inputState.guessing==0 ) {
                              arith_factor_AST = (AST)currentAST.root;
                              arith_factor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(UPLUS,"UPLUS")).add(arith_factor_AST));
                              currentAST.root = arith_factor_AST;
                              currentAST.child = arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                                    arith_factor_AST.getFirstChild() :
arith_factor_AST;
                              currentAST.advanceChildToEnd();
                        }
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case MINUS:
                  {
                        match(MINUS);
                        r_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        if ( inputState.guessing==0 ) {
                              arith_factor_AST = (AST)currentAST.root;
                              arith_factor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(UMINUS,"UMINUS")).add(arith_factor_AST));
                              currentAST.root = arith_factor_AST;
                              currentAST.child = arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                                    arith_factor_AST.getFirstChild() :
arith_factor_AST;
                              currentAST.advanceChildToEnd();
                        }
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case DPLUS:
                  {
                        match(DPLUS);
                        l_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        if ( inputState.guessing==0 ) {
                              arith_factor_AST = (AST)currentAST.root;
```

```
                              arith_factor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(INCREMENT,"INCREMENT")).add(arith_factor_AST)
);
                              currentAST.root = arith_factor_AST;
                              currentAST.child = arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                                    arith_factor_AST.getFirstChild() :
arith_factor_AST;
                              currentAST.advanceChildToEnd();
                        }
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case DMINUS:
                  {
                        match(DMINUS);
                        l_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        if ( inputState.guessing==0 ) {
                              arith_factor_AST = (AST)currentAST.root;
                              arith_factor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(DECREMENT,"DECREMENT")).add(arith_factor_AST)
);
                              currentAST.root = arith_factor_AST;
                              currentAST.child = arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                                    arith_factor_AST.getFirstChild() :
arith_factor_AST;
                              currentAST.advanceChildToEnd();
                        }
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case NOT:
                  {
                        match(NOT);
                        r_value();
                        astFactory.addASTChild(currentAST, returnAST);
                        if ( inputState.guessing==0 ) {
                              arith_factor_AST = (AST)currentAST.root;
                              arith_factor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NEGATION,"NEGATION")).add(arith_factor_AST));
                              currentAST.root = arith_factor_AST;
                              currentAST.child = arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                                    arith_factor_AST.getFirstChild() :
arith_factor_AST;
                              currentAST.advanceChildToEnd();
                        }
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case LPAREN:
                  case ID:
                  case NUMBER:
                  {
                        r_value();
```

```
                        astFactory.addASTChild(currentAST, returnAST);
                        arith_factor_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1), getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  if (inputState.guessing==0) {
                        reportError(ex);
                        recover(ex,_tokenSet_7);
                  } else {
                    throw ex;
                  }
            }
            returnAST = arith_factor_AST;
      }

      public final void r_value() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST r_value_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case NUMBER:
                  {
                        AST tmp67_AST = null;
                        tmp67_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp67_AST);
                        match(NUMBER);
                        r_value_AST = (AST)currentAST.root;
                        break;
                  }
                  case LPAREN:
                  {
                        match(LPAREN);
                        expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RPAREN);
                        r_value_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                        if ((LA(1)==ID) && (_tokenSet_7.member(LA(2)))) {
                              l_value();
                              astFactory.addASTChild(currentAST, returnAST);
                              r_value_AST = (AST)currentAST.root;
                        }
                        else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
                              func_call();
                              astFactory.addASTChild(currentAST, returnAST);
```

```
                        r_value_AST = (AST)currentAST.root;
                  }
            else {
                  throw new NoViableAltException(LT(1), getFilename());
            }
            }
      }
      catch (RecognitionException ex) {
            if (inputState.guessing==0) {
                  reportError(ex);
                  recover(ex,_tokenSet_7);
            } else {
              throw ex;
            }
      }
      returnAST = r_value_AST;
}


public static final String[] _tokenNames = {
      "<0>",
      "EOF",
      "<2>",
      "NULL_TREE_LOOKAHEAD",
      "ALPHA",
      "DIGIT",
      "WS",
      "NL",
      "COMMENT",
      "LPAREN",
      "RPAREN",
      "MULT",
      "PLUS",
      "MINUS",
      "DIV",
      "MOD",
      "COMMA",
      "SEMI",
      "LBRACE",
      "RBRACE",
      "ASGN",
      "PLUSEQ",
      "MINUSEQ",
      "MULTEQ",
      "DIVEQ",
      "MODEQ",
      "GE",
      "LE",
      "GT",
      "LT",
      "EQ",
      "NEQ",
      "DPLUS",
      "DMINUS",
      "DBAR",
      "DAMP",
      "NOT",
```

```
        "ID",
        "NUMBER",
        "STRING",
        "STATEMENT",
        "FUNC_CALL",
        "EXPR_LIST",
        "VAR_LIST",
        "UPLUS",
        "UMINUS",
        "INCREMENT",
        "DECREMENT",
        "NEGATION",
        "\"for\"",
        "\"to\"",
        "\"if\"",
        "\"else\"",
        "\"return\"",
        "\"include\"",
        "\"func\""
};

protected void buildTokenTypeASTClassMap() {
        tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
        long[] data = { 2L, 0L};
        return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
        long[] data = { 29836497855447040L, 0L};
        return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
        long[] data = { 70368894502305794L, 0L};
        return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
        long[] data = { 65865294874411010L, 0L};
        return data;
}
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
private static final long[] mk_tokenSet_4() {
        long[] data = { 70369787921707522L, 0L};
        return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
        long[] data = { 1125899907040256L, 0L};
        return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
        long[] data = { 1125955740630016L, 0L};
```

```
	return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
	long[] data = { 1125955674569728L, 0L};
	return data;
}
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
	long[] data = { 1024L, 0L};
	return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
	long[] data = { 1125917086909440L, 0L};
	return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
	long[] data = { 1125951446647808L, 0L};
	return data;
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
	long[] data = { 1125955674506240L, 0L};
	return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());
private static final long[] mk_tokenSet_12() {
	long[] data = { 1125955674518528L, 0L};
	return data;
}
public static final BitSet _tokenSet_12 = new BitSet(mk_tokenSet_12());

}
```

```
/***********************************************************
 * AnimoAntlrTokenTypes.java: Created by Antlr
 ***********************************************************/

// $ANTLR 2.7.5 (20050128): "grammar.g" -> "AnimoAntlrLexer.java"$

public interface AnimoAntlrTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int ALPHA = 4;
	int DIGIT = 5;
	int WS = 6;
	int NL = 7;
	int COMMENT = 8;
	int LPAREN = 9;
	int RPAREN = 10;
	int MULT = 11;
	int PLUS = 12;
	int MINUS = 13;
	int DIV = 14;
	int MOD = 15;
	int COMMA = 16;
	int SEMI = 17;
	int LBRACE = 18;
	int RBRACE = 19;
	int ASGN = 20;
	int PLUSEQ = 21;
	int MINUSEQ = 22;
	int MULTEQ = 23;
	int DIVEQ = 24;
	int MODEQ = 25;
	int GE = 26;
	int LE = 27;
	int GT = 28;
	int LT = 29;
	int EQ = 30;
	int NEQ = 31;
	int DPLUS = 32;
	int DMINUS = 33;
	int DBAR = 34;
	int DAMP = 35;
	int NOT = 36;
	int ID = 37;
	int NUMBER = 38;
	int STRING = 39;
	int STATEMENT = 40;
	int FUNC_CALL = 41;
	int EXPR_LIST = 42;
	int VAR_LIST = 43;
	int UPLUS = 44;
	int UMINUS = 45;
	int INCREMENT = 46;
	int DECREMENT = 47;
	int NEGATION = 48;
	int LITERAL_for = 49;
	int LITERAL_to = 50;
	int LITERAL_if = 51;
```

```
        int LITERAL_else = 52;
        int LITERAL_return = 53;
        int LITERAL_include = 54;
        int LITERAL_func = 55;
    }
```

```
/***********************************************
 * AnimoAntlrWalker.java: Created by Antlr
 ***********************************************/

// $ANTLR 2.7.5 (20050128): "walker.g" -> "AnimoAntlrWalker.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
import java.util.*;


public class AnimoAntlrWalker extends antlr.TreeParser       implements
AnimoAntlrWalkerTokenTypes
 {

    static AnimoDataType null_data = new AnimoDataType( "NULL" );
    AnimoCompiler cmpl = new AnimoCompiler();
public AnimoAntlrWalker() {
      tokenNames = _tokenNames;
}

      public final  AnimoDataType  expr(AST _t) throws RecognitionException {
             AnimoDataType r ;

            AST expr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
            AST num = null;
            AST id = null;
            AST forbody = null;
            AST thenp = null;
            AST elsep = null;
            AST stmt = null;
            AST fname = null;
            AST fbody = null;

            AnimoDataType a, b, c;
            AnimoDataType[] x = null;
            String s = null;
            String[] sx = null;
            r = null_data;


            try {      // for error handling
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case DBAR:
                  {
```

```
      AST __t2 = _t;
      AST tmp1_AST_in = (AST)_t;
      match(_t,DBAR);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t2;
      _t = _t.getNextSibling();
      r = a.or(b);
      break;
}
case DAMP:
{
      AST __t3 = _t;
      AST tmp2_AST_in = (AST)_t;
      match(_t,DAMP);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t3;
      _t = _t.getNextSibling();
      r = a.and(b);
      break;
}
case NEGATION:
{
      AST __t4 = _t;
      AST tmp3_AST_in = (AST)_t;
      match(_t,NEGATION);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      _t = __t4;
      _t = _t.getNextSibling();
      r = a.not();
      break;
}
case INCREMENT:
{
      AST __t5 = _t;
      AST tmp4_AST_in = (AST)_t;
      match(_t,INCREMENT);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      _t = __t5;
      _t = _t.getNextSibling();
      r = a.incr();
      break;
}
case DECREMENT:
{
      AST __t6 = _t;
```

```
      AST tmp5_AST_in = (AST)_t;
      match(_t,DECREMENT);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      _t = __t6;
      _t = _t.getNextSibling();
      r = a.decr();
      break;
}
case GE:
{
      AST __t7 = _t;
      AST tmp6_AST_in = (AST)_t;
      match(_t,GE);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t7;
      _t = _t.getNextSibling();
      r = a.ge( b );
      break;
}
case LE:
{
      AST __t8 = _t;
      AST tmp7_AST_in = (AST)_t;
      match(_t,LE);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t8;
      _t = _t.getNextSibling();
      r = a.le( b );
      break;
}
case GT:
{
      AST __t9 = _t;
      AST tmp8_AST_in = (AST)_t;
      match(_t,GT);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t9;
      _t = _t.getNextSibling();
      r = a.gt( b );
      break;
}
case LT:
{
```

```
        AST __t10 = _t;
        AST tmp9_AST_in = (AST)_t;
        match(_t,LT);
        _t = _t.getFirstChild();
        a=expr(_t);
        _t = _retTree;
        b=expr(_t);
        _t = _retTree;
        _t = __t10;
        _t = _t.getNextSibling();
        r = a.lt( b );
        break;
    }
    case EQ:
    {
        AST __t11 = _t;
        AST tmp10_AST_in = (AST)_t;
        match(_t,EQ);
        _t = _t.getFirstChild();
        a=expr(_t);
        _t = _retTree;
        b=expr(_t);
        _t = _retTree;
        _t = __t11;
        _t = _t.getNextSibling();
        r = a.eq( b );
        break;
    }
    case NEQ:
    {
        AST __t12 = _t;
        AST tmp11_AST_in = (AST)_t;
        match(_t,NEQ);
        _t = _t.getFirstChild();
        a=expr(_t);
        _t = _retTree;
        b=expr(_t);
        _t = _retTree;
        _t = __t12;
        _t = _t.getNextSibling();
        r = a.ne( b );
        break;
    }
    case PLUS:
    {
        AST __t13 = _t;
        AST tmp12_AST_in = (AST)_t;
        match(_t,PLUS);
        _t = _t.getFirstChild();
        a=expr(_t);
        _t = _retTree;
        b=expr(_t);
        _t = _retTree;
        _t = __t13;
        _t = _t.getNextSibling();
        r = a.plus( b );
        break;
```

```
}
case MINUS:
{
      AST __t14 = _t;
      AST tmp13_AST_in = (AST)_t;
      match(_t,MINUS);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t14;
      _t = _t.getNextSibling();
      r = a.minus( b );
      break;
}
case MULT:
{
      AST __t15 = _t;
      AST tmp14_AST_in = (AST)_t;
      match(_t,MULT);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t15;
      _t = _t.getNextSibling();
      r = a.times( b );
      break;
}
case DIV:
{
      AST __t16 = _t;
      AST tmp15_AST_in = (AST)_t;
      match(_t,DIV);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t16;
      _t = _t.getNextSibling();
      r = a.divby( b );
      break;
}
case MOD:
{
      AST __t17 = _t;
      AST tmp16_AST_in = (AST)_t;
      match(_t,MOD);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t17;
```

```
      _t = _t.getNextSibling();
      r = a.modulus( b );
      break;
}
case UPLUS:
{
      AST __t18 = _t;
      AST tmp17_AST_in = (AST)_t;
      match(_t,UPLUS);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      _t = __t18;
      _t = _t.getNextSibling();
      r = a;
      break;
}
case UMINUS:
{
      AST __t19 = _t;
      AST tmp18_AST_in = (AST)_t;
      match(_t,UMINUS);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      _t = __t19;
      _t = _t.getNextSibling();
      r = a.uminus();
      break;
}
case PLUSEQ:
{
      AST __t20 = _t;
      AST tmp19_AST_in = (AST)_t;
      match(_t,PLUSEQ);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t20;
      _t = _t.getNextSibling();
      r = a.add( b );
      break;
}
case MINUSEQ:
{
      AST __t21 = _t;
      AST tmp20_AST_in = (AST)_t;
      match(_t,MINUSEQ);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t21;
      _t = _t.getNextSibling();
```

```
      r = a.sub( b );
      break;
}
case MULTEQ:
{
      AST __t22 = _t;
      AST tmp21_AST_in = (AST)_t;
      match(_t,MULTEQ);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t22;
      _t = _t.getNextSibling();
      r = a.mul( b );
      break;
}
case DIVEQ:
{
      AST __t23 = _t;
      AST tmp22_AST_in = (AST)_t;
      match(_t,DIVEQ);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t23;
      _t = _t.getNextSibling();
      r = a.div( b );
      break;
}
case MODEQ:
{
      AST __t24 = _t;
      AST tmp23_AST_in = (AST)_t;
      match(_t,MODEQ);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
      _t = _retTree;
      _t = __t24;
      _t = _t.getNextSibling();
      r = a.rem( b );
      break;
}
case ASGN:
{
      AST __t25 = _t;
      AST tmp24_AST_in = (AST)_t;
      match(_t,ASGN);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      b=expr(_t);
```

```
	_t = _retTree;
	_t = __t25;
	_t = _t.getNextSibling();
	r = cmpl.assign( a, b );
	break;
}
case FUNC_CALL:
{
	AST __t26 = _t;
	AST tmp25_AST_in = (AST)_t;
	match(_t,FUNC_CALL);
	_t = _t.getFirstChild();
	a=fexpr(_t);
	_t = _retTree;
	{
	if (_t==null) _t=ASTNULL;
	switch ( _t.getType()) {
	case MULT:
	case PLUS:
	case MINUS:
	case DIV:
	case MOD:
	case ASGN:
	case PLUSEQ:
	case MINUSEQ:
	case MULTEQ:
	case DIVEQ:
	case MODEQ:
	case GE:
	case LE:
	case GT:
	case LT:
	case EQ:
	case NEQ:
	case DBAR:
	case DAMP:
	case ID:
	case NUMBER:
	case STATEMENT:
	case FUNC_CALL:
	case EXPR_LIST:
	case UPLUS:
	case UMINUS:
	case INCREMENT:
	case DECREMENT:
	case NEGATION:
	case LITERAL_for:
	case LITERAL_if:
	case LITERAL_return:
	case LITERAL_include:
	case LITERAL_func:
	{
		x=mexpr(_t);
		_t = _retTree;
		break;
	}
	case 3:
```

```
                        {
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(_t);
                        }
                        }
                        }
                        _t = __t26;
                        _t = _t.getNextSibling();

                        r = cmpl.funcInvoke( this, a, x );

                        break;
                  }
                  case NUMBER:
                  {
                        num = (AST)_t;
                        match(_t,NUMBER);
                        _t = _t.getNextSibling();
                        r = cmpl.getNumber( num.getText() );
                        break;
                  }
                  case ID:
                  {
                        id = (AST)_t;
                        match(_t,ID);
                        _t = _t.getNextSibling();
                        r = cmpl.getValue( id.getText() );
                        break;
                  }
                  case LITERAL_for:
                  {
                        AST __t28 = _t;
                        AST tmp26_AST_in = (AST)_t;
                        match(_t,LITERAL_for);
                        _t = _t.getFirstChild();
                        a=expr(_t);
                        _t = _retTree;
                        b=expr(_t);
                        _t = _retTree;
                        c=expr(_t);
                        _t = _retTree;
                        forbody = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenException();
                        _t = _t.getNextSibling();
                        _t = __t28;
                        _t = _t.getNextSibling();

                        if (b.typename().equals("unknown"))
                        return b.error( "for: uninitialized variable in lower
bound " );
                        if (c.typename().equals("unknown"))
                        return c.error( "for: uninitialized variable in upper
bound " );
```

```
                        if ( !( b instanceof AnimoFloatingPoint ) )
                        return b.error( "for: lower bound is not an integer" );
                        if ( !( c instanceof AnimoFloatingPoint ) )
                        return c.error( "for: upper bound is not an integer" );

                        double bVar = ((AnimoFloatingPoint)b).var;
                        if (bVar != Math.floor(bVar))
                        return b.error( "for: lower bound " + bVar + " is not an
integer" );

                        double cVar = ((AnimoFloatingPoint)c).var;
                        if (cVar != Math.floor(cVar))
                        return c.error( "for: upper bound " + cVar + " is not an
integer" );

                        cmpl.enterScope();
                        cmpl.assign(a, b);
                        AnimoDataType counter = cmpl.getValue( a.name );
                        while (((AnimoFloatingPoint)counter).var <=
                        ((AnimoFloatingPoint)c).var)
                        {
                        r = expr(forbody);
                        ((AnimoFloatingPoint)counter).var++;
                        }
                        ((AnimoFloatingPoint)counter).var--;
                        cmpl.leaveScope();

                        break;
                  }
                  case LITERAL_if:
                  {
                        AST __t29 = _t;
                        AST tmp27_AST_in = (AST)_t;
                        match(_t,LITERAL_if);
                        _t = _t.getFirstChild();
                        a=expr(_t);
                        _t = _retTree;
                        thenp = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenException();
                        _t = _t.getNextSibling();
                        {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ALPHA:
                        case DIGIT:
                        case WS:
                        case NL:
                        case COMMENT:
                        case LPAREN:
                        case RPAREN:
                        case MULT:
                        case PLUS:
                        case MINUS:
                        case DIV:
                        case MOD:
                        case COMMA:
                        case SEMI:
```

```
                        case LBRACE:
                        case RBRACE:
                        case ASGN:
                        case PLUSEQ:
                        case MINUSEQ:
                        case MULTEQ:
                        case DIVEQ:
                        case MODEQ:
                        case GE:
                        case LE:
                        case GT:
                        case LT:
                        case EQ:
                        case NEQ:
                        case DPLUS:
                        case DMINUS:
                        case DBAR:
                        case DAMP:
                        case NOT:
                        case ID:
                        case NUMBER:
                        case STRING:
                        case STATEMENT:
                        case FUNC_CALL:
                        case EXPR_LIST:
                        case VAR_LIST:
                        case UPLUS:
                        case UMINUS:
                        case INCREMENT:
                        case DECREMENT:
                        case NEGATION:
                        case LITERAL_for:
                        case LITERAL_to:
                        case LITERAL_if:
                        case LITERAL_else:
                        case LITERAL_return:
                        case LITERAL_include:
                        case LITERAL_func:
                        {
                              elsep = (AST)_t;
                              if ( _t==null ) throw new
MismatchedTokenException();
                              _t = _t.getNextSibling();
                              break;
                        }
                        case 3:
                        {
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(_t);
                        }
                        }
                        }
                        _t = __t29;
                        _t = _t.getNextSibling();
```

```
                        if ( a.typename().equals("unknown") )
                        return a.error( "if: use of uninitialized value in " +
                        "expression" );
                        if ( !( a instanceof AnimoFloatingPoint ) )
                        return a.error( "if: expression is not a floating point"
);
                        cmpl.enterScope();
                        if ( ((AnimoFloatingPoint)a).var != 0.0)
                        r = expr( thenp );
                        else if ( null != elsep )
                        r = expr( elsep );
                        cmpl.leaveScope();

                        break;
                  }
                  case STATEMENT:
                  {
                        AST __t31 = _t;
                        AST tmp28_AST_in = (AST)_t;
                        match(_t,STATEMENT);
                        _t = _t.getFirstChild();
                        {
                        _loop33:
                        do {
                              if (_t==null) _t=ASTNULL;
                              if (((_t.getType() >= ALPHA && _t.getType() <=
LITERAL_func))) {
                                    stmt = (AST)_t;
                                    if ( _t==null ) throw new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    if ( cmpl.canProceed() ) r = expr(stmt);
                              }
                              else {
                                    break _loop33;
                              }

                        } while (true);
                        }
                        _t = __t31;
                        _t = _t.getNextSibling();
                        break;
                  }
                  case LITERAL_return:
                  {
                        AST __t34 = _t;
                        AST tmp29_AST_in = (AST)_t;
                        match(_t,LITERAL_return);
                        _t = _t.getFirstChild();
                        {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case MULT:
                        case PLUS:
                        case MINUS:
                        case DIV:
```

```
	case MOD:
	case ASGN:
	case PLUSEQ:
	case MINUSEQ:
	case MULTEQ:
	case DIVEQ:
	case MODEQ:
	case GE:
	case LE:
	case GT:
	case LT:
	case EQ:
	case NEQ:
	case DBAR:
	case DAMP:
	case ID:
	case NUMBER:
	case STATEMENT:
	case FUNC_CALL:
	case UPLUS:
	case UMINUS:
	case INCREMENT:
	case DECREMENT:
	case NEGATION:
	case LITERAL_for:
	case LITERAL_if:
	case LITERAL_return:
	case LITERAL_include:
	case LITERAL_func:
	{
		a=expr(_t);
		_t = _retTree;
		r = cmpl.rvalue( a );
		break;
	}
	case 3:
	{
		break;
	}
	default:
	{
		throw new NoViableAltException(_t);
	}
	}
	}
	_t = __t34;
	_t = _t.getNextSibling();
	cmpl.setReturn();
	break;
}
case LITERAL_func:
{
	AST __t36 = _t;
	AST tmp30_AST_in = (AST)_t;
	match(_t,LITERAL_func);
	_t = _t.getFirstChild();
	fname = (AST)_t;
```

```
                  match(_t,ID);
                  _t = _t.getNextSibling();
                  sx=vlist(_t);
                  _t = _retTree;
                  fbody = (AST)_t;
                  if ( _t==null ) throw new MismatchedTokenException();
                  _t = _t.getNextSibling();
                  _t = __t36;
                  _t = _t.getNextSibling();
                  cmpl.funcRegister( fname.getText(), sx, fbody );
                  break;
            }
            case LITERAL_include:
            {
                  AST __t37 = _t;
                  AST tmp31_AST_in = (AST)_t;
                  match(_t,LITERAL_include);
                  _t = _t.getFirstChild();
                  s=string(_t);
                  _t = _retTree;
                  _t = __t37;
                  _t = _t.getNextSibling();
                  cmpl.include( s );
                  break;
            }
            default:
            {
                  throw new NoViableAltException(_t);
            }
            }
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      _retTree = _t;
      return r ;
}

public final  AnimoDataType  fexpr(AST _t) throws RecognitionException {
       AnimoDataType r ;

      AST fexpr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
      AST id = null;

      r = null;


      try {      // for error handling
            id = (AST)_t;
            match(_t,ID);
            _t = _t.getNextSibling();
            r = cmpl.getFunction(id.getText());
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
```

```
	}
	_retTree = _t;
	return r ;
}

public final  AnimoDataType[]  mexpr(AST _t) throws RecognitionException {
	 AnimoDataType[] rv ;

	AST mexpr_AST_in = (_t == ASTNULL) ? null : (AST)_t;

	AnimoDataType a;
	rv = null;
	Vector v;


	try {	// for error handling
		if (_t==null) _t=ASTNULL;
		switch ( _t.getType()) {
		case EXPR_LIST:
		{
			AST __t39 = _t;
			AST tmp32_AST_in = (AST)_t;
			match(_t,EXPR_LIST);
			_t = _t.getFirstChild();
			v = new Vector();
			{
			_loop41:
			do {
				if (_t==null) _t=ASTNULL;
				if ((_tokenSet_0.member(_t.getType()))) {
					a=expr(_t);
					_t = _retTree;
					v.add( a );
				}
				else {
					break _loop41;
				}

			} while (true);
			}
			_t = __t39;
			_t = _t.getNextSibling();
			rv = cmpl.convertExprList( v );
			break;
		}
		case MULT:
		case PLUS:
		case MINUS:
		case DIV:
		case MOD:
		case ASGN:
		case PLUSEQ:
		case MINUSEQ:
		case MULTEQ:
		case DIVEQ:
		case MODEQ:
		case GE:
```

```
            case LE:
            case GT:
            case LT:
            case EQ:
            case NEQ:
            case DBAR:
            case DAMP:
            case ID:
            case NUMBER:
            case STATEMENT:
            case FUNC_CALL:
            case UPLUS:
            case UMINUS:
            case INCREMENT:
            case DECREMENT:
            case NEGATION:
            case LITERAL_for:
            case LITERAL_if:
            case LITERAL_return:
            case LITERAL_include:
            case LITERAL_func:
            {
                  a=expr(_t);
                  _t = _retTree;
                  rv = new AnimoDataType[1]; rv[0] = a;
                  break;
            }
            default:
            {
                  throw new NoViableAltException(_t);
            }
            }
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      _retTree = _t;
      return rv ;
}

public final  String[]  vlist(AST _t) throws RecognitionException {
       String[] sv ;

      AST vlist_AST_in = (_t == ASTNULL) ? null : (AST)_t;
      AST s = null;

      Vector v;
      sv = null;


      try {      // for error handling
            AST __t43 = _t;
            AST tmp33_AST_in = (AST)_t;
            match(_t,VAR_LIST);
            _t = _t.getFirstChild();
            v = new Vector();
```

```
            {
            _loop45:
            do {
                  if (_t==null) _t=ASTNULL;
                  if ((_t.getType()==ID)) {
                        s = (AST)_t;
                        match(_t,ID);
                        _t = _t.getNextSibling();
                        v.add( s.getText() );
                  }
                  else {
                        break _loop45;
                  }

            } while (true);
            }
            _t = __t43;
            _t = _t.getNextSibling();
            sv = cmpl.convertVarList( v );
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      _retTree = _t;
      return sv ;
}

public final  String  string(AST _t) throws RecognitionException {
       String s ;

      AST string_AST_in = (_t == ASTNULL) ? null : (AST)_t;
      AST str = null;

      s = null;


      try {      // for error handling
            str = (AST)_t;
            match(_t,STRING);
            _t = _t.getNextSibling();
            s = str.getText();
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      _retTree = _t;
      return s ;
}


public static final String[] _tokenNames = {
      "<0>",
      "EOF",
      "<2>",
      "NULL_TREE_LOOKAHEAD",
```

```
      "ALPHA",
      "DIGIT",
      "WS",
      "NL",
      "COMMENT",
      "LPAREN",
      "RPAREN",
      "MULT",
      "PLUS",
      "MINUS",
      "DIV",
      "MOD",
      "COMMA",
      "SEMI",
      "LBRACE",
      "RBRACE",
      "ASGN",
      "PLUSEQ",
      "MINUSEQ",
      "MULTEQ",
      "DIVEQ",
      "MODEQ",
      "GE",
      "LE",
      "GT",
      "LT",
      "EQ",
      "NEQ",
      "DPLUS",
      "DMINUS",
      "DBAR",
      "DAMP",
      "NOT",
      "ID",
      "NUMBER",
      "STRING",
      "STATEMENT",
      "FUNC_CALL",
      "EXPR_LIST",
      "VAR_LIST",
      "UPLUS",
      "UMINUS",
      "INCREMENT",
      "DECREMENT",
      "NEGATION",
      "\"for\"",
      "\"to\"",
      "\"if\"",
      "\"else\"",
      "\"return\"",
      "\"include\"",
      "\"func\""
};

private static final long[] mk_tokenSet_0() {
      long[] data = { 66414269003003904L, 0L};
      return data;
```

```
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
}
```

```
/***********************************************************
 * AnimoAntlrWalkerTokenTypes.java: Created by Antlr
 ***********************************************************/

// $ANTLR 2.7.5 (20050128): "walker.g" -> "AnimoAntlrWalker.java"$

public interface AnimoAntlrWalkerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int ALPHA = 4;
	int DIGIT = 5;
	int WS = 6;
	int NL = 7;
	int COMMENT = 8;
	int LPAREN = 9;
	int RPAREN = 10;
	int MULT = 11;
	int PLUS = 12;
	int MINUS = 13;
	int DIV = 14;
	int MOD = 15;
	int COMMA = 16;
	int SEMI = 17;
	int LBRACE = 18;
	int RBRACE = 19;
	int ASGN = 20;
	int PLUSEQ = 21;
	int MINUSEQ = 22;
	int MULTEQ = 23;
	int DIVEQ = 24;
	int MODEQ = 25;
	int GE = 26;
	int LE = 27;
	int GT = 28;
	int LT = 29;
	int EQ = 30;
	int NEQ = 31;
	int DPLUS = 32;
	int DMINUS = 33;
	int DBAR = 34;
	int DAMP = 35;
	int NOT = 36;
	int ID = 37;
	int NUMBER = 38;
	int STRING = 39;
	int STATEMENT = 40;
	int FUNC_CALL = 41;
	int EXPR_LIST = 42;
	int VAR_LIST = 43;
	int UPLUS = 44;
	int UMINUS = 45;
	int INCREMENT = 46;
	int DECREMENT = 47;
	int NEGATION = 48;
	int LITERAL_for = 49;
	int LITERAL_to = 50;
	int LITERAL_if = 51;
```

```
        int LITERAL_else = 52;
        int LITERAL_return = 53;
        int LITERAL_include = 54;
        int LITERAL_func = 55;
}
```

```
//***************************************************************************
// AnimoBVHJoint.java
//
// Represents a BVH joint in ANIMO.  Not much info about a joint is defined
// here...use AnimoMatrix to get/set a joint's rotation/position in a motion
// matrix.
//
// Last Modified: 12/16/2005    By: Josh Poritz
//***************************************************************************

import java.io.PrintWriter;

public class AnimoBVHJoint extends AnimoDataType
{
    boolean isRoot = false;
    String jointName = null;

    static String myType = "bvhjoint";

    public AnimoBVHJoint(String name, String jointName, boolean isRoot) {
        super(name);
      this.jointName = jointName;
      this.isRoot = isRoot;
    }

    public String typename() {
        return myType;
    }

    public AnimoDataType copy() {
        return new AnimoBVHJoint(name, jointName, isRoot);
    }

    public void print(PrintWriter w) {
      w.println(jointName);
    }

    public boolean isRoot() {
      return isRoot;
    }
}
```

```
//**************************************************************************
// AnimoCompiler.java
//
// Manipulates symbol tables, implements scoping, includes all functionality
// for the walker.
//
// Author: Natasha Shamis
//**************************************************************************


import java.util.*;
import java.io.*;
import antlr.*;
import antlr.collections.AST;

public class AnimoCompiler
{
    private AnimoSymbolTable floats;
    private static AnimoSymbolTable joints = new AnimoSymbolTable("BVHJoint");
    private static AnimoSymbolTable funcs = new AnimoSymbolTable("Functions");
    private AnimoSymbolTable top;//floating point table at top of scope stack

    private Stack floats_scope;

    // List of included ANIMO files
    private static Hashtable animoFiles = new Hashtable();

    // Motion matrix
    private static AnimoMatrix matrix = new AnimoMatrix();

    private final static int fc_none = 0;
    private final static int fc_return = 1;

    private int control = fc_none;
    private int level;//level of scope (level 1 is global scope)
    private static boolean hasLoadedBVH = false;

    public AnimoCompiler() {
      floats = new AnimoSymbolTable("FloatingPoint");
      top = floats;
      floats_scope = new Stack();
      floats_scope.push(floats);
      level = 1;//global scope
      AnimoInternalFunction.register(funcs, floats);
    }

    public AnimoDataType[] convertExprList(Vector v) {
        /* Note: expr list can be empty */
        AnimoDataType[] x = new AnimoDataType[v.size()];
        for (int i = 0; i < x.length; i++)
            x[i] = (AnimoDataType) v.elementAt(i);
        return x;
    }

   public static String[] convertVarList(Vector v) {
        /* Note: var list can be empty */
        String[] str = new String[v.size()];
```

```
        for (int i = 0; i < str.length; i++)
      {
            str[i] = (String) v.elementAt(i);

          // Disallow use of BVH joint in prototype
          if (joints.get(str[i]) != null)
            throw new AnimoException("Illegal use to BVH joint in " +
                                "function prototype: " + str[i]);
      }
        return str;
    }

    public AnimoFloatingPoint getNumber(String num) {
      try {
           return new AnimoFloatingPoint(Double.parseDouble(num));
      }

      catch (NumberFormatException e) {
         throw new AnimoException("Parse error at '" + num + "': not a floating
point\n");
      }
    }

    public AnimoDataType getValue(String key) {
      AnimoDataType x;
      if ((x = joints.get(key)) == null)
          if ((x = top.get(key)) == null)
            return new AnimoDataType(key);
      return x;
    }

    public AnimoDataType getFunction(String key) {
      AnimoDataType x;
      if ((x = funcs.get(key)) != null)
          return x;
      throw new AnimoException("Can't invoke undefined function: " + key +
                          "()");
    }

    public AnimoDataType rvalue(AnimoDataType a) {
        if (a.name == null)
            return a;
        return a.copy();
    }

    public AnimoDataType assign(AnimoDataType a, AnimoDataType b ) {
        if (a.name != null) {
          if (!isInitialized(b))
            throw new AnimoException("Use of unitialized value " +
                                (b.name != null ? "\"" + b.name + "\""
                                : "<?>") + " on right-hand side of " +
                                "assignment to \"" + a.name + "\"");

          AnimoDataType x = rvalue(b);

          // Assigning to a BVH joint name is forbidden
          if (x instanceof AnimoBVHJoint)
```

```
          {
            AnimoBVHJoint j = (AnimoBVHJoint) joints.get(a.name);
            if (j != null && a.name.equals(j.jointName))
                throw new AnimoException("Illegal assignment to BVH " +
                                         "joint: " + a.name);
          }

          x.setName(a.name);
          varRegister(x);
            return x;
        }
        return a.badTypeError( b, "=" );
    }

    //Add scope to stack
    public void enterScope() {
      level++;
        AnimoSymbolTable tab = new
AnimoSymbolTable("FloatingPoint"+String.valueOf(level), top);
      //parent of new table is set to table previously at top of stack
        floats_scope.push(tab);
        top = tab;
    }

    //Remove most recent scope from stack
    public void leaveScope() {
        AnimoSymbolTable tab = (AnimoSymbolTable)floats_scope.pop();
      top = tab.getParent();//table in previous scope is back to top of stack
      level--;
    }

    public AnimoDataType funcInvoke(AnimoAntlrWalker walker, AnimoDataType func,
AnimoDataType[] params )
            throws antlr.RecognitionException {
        // func must be an existing function
        if (!(func instanceof AnimoFunction))
            return func.error("Not a function\n");

      // Ensure the arguments are initialized
      if (params != null)
          for (int i = 0; i < params.length; i++)
            if (!isInitialized(params[i]))
                throw new AnimoException("Use of uninitialized variable " +
                                         "\"" + params[i].name + "\" in " +
                                         "invocation of " +
                                         (func.name != null ?
                                         func.name : "internal function"));

        // Is this function an internal function?
        // Names of formal args are not necessary for internal functions.
        if (((AnimoFunction)func).isInternal())
          return AnimoInternalFunction.run(funcs, matrix,
((AnimoFunction)func).getInternalId(), params);

        // otherwise check numbers of actual and formal arguments
        String[] args = ((AnimoFunction)func).getArgs();
      if (params == null)
```

```
  {
      if (args.length != 0)
        return func.error("Mismatching number of parameters");
  }
  else if (args.length != params.length)
      return func.error("Mismatching number of parameters");

    // Enter a new scope
  enterScope();

    // assign actual parameters to formal arguments
    for (int i = 0; i < args.length; i++) {
        AnimoDataType d = rvalue(params[i]);
        d.setName(args[i]);
        top.put(args[i], d);
    }

    // call the function body
    AnimoDataType r = walker.expr(((AnimoFunction)func).getBody());

  if (r == null)
      r = new AnimoDataType("NULL");

  // if a return was called
    if (control == fc_return)
        tryResetFlowControl();

    // Leave this scope
    leaveScope();

    return r;
}

public void funcRegister(String name, String[] args, AST body) {
  if (funcs.get(name) != null)
      throw new AnimoException("Illegal re-definition of function: " +
                           name + "()");
    funcs.put(name, new AnimoFunction(name, args, body, funcs));
}

public void varRegister(AnimoDataType var) {
  if (var instanceof AnimoFloatingPoint)
  {
      if (joints.get(var.name) == null)
        top.put(var.name, var);
      else
        throw new AnimoException("Can't initialize variable \"" +
                             var.name + "\": A BVH joint with " +
                             "that name already exists.");
  }
  else if (var instanceof AnimoBVHJoint)
  {
      if (top.get(var.name) == null)
        joints.put(var.name, var);
      else
        throw new AnimoException("Can't register BVH joint \"" +
                             var.name + "\": A floating point " +
```

```
                              "with that name lready exists.");
      }
      else
          throw new AnimoException("Invalid data format for variable " +
                               "registration: " + var.name);
    }

    public void setReturn() {
        control = fc_return;
    }

    public void tryResetFlowControl() {
        control = fc_none;
    }

    public boolean canProceed() {
        return control == fc_none;
    }

    public boolean isInitialized(AnimoDataType x) {
      return (x != null && !x.typename().equals("unknown"));
    }

    public AnimoDataType include(String file) {
            try
            {
            if (file.endsWith(".bvh") || file.endsWith(".sbvh")) {
                if (!hasLoadedBVH) {
                  matrix.loadBVHfile(file);

                  // Add BVH joints to BVHJoint symbol table
                  Set BVHJoints = matrix.getAllJoints();
                  Iterator it = BVHJoints.iterator();
                  while (it.hasNext())
                  {
                      String jointName;
                      if ((jointName = (String) it.next()) == null)
                        continue;
                      varRegister(new AnimoBVHJoint(jointName,
                                                    jointName,
                              matrix.getJoint(jointName).isRoot()));
                  }

                  hasLoadedBVH = true;
                  return null;
                }
                else
                  throw new AnimoException("Inclusion of multiple BVH files not
supported.");
            }

            if (animoFiles.get(file) != null)
                throw new AnimoException("Can't include file \"" + file +
                                    "\": Multiple inclusions of " +
                                    "the same file are not " +
                                    "permitted");
            animoFiles.put(file, 1);
```

```
                InputStream input = (InputStream) new FileInputStream(file);
                AnimoAntlrLexer lexer = new AnimoAntlrLexer(input);
                AnimoAntlrParser parser = new AnimoAntlrParser(lexer);
                parser.program();
                CommonAST tree = (CommonAST)parser.getAST();
                AnimoAntlrWalker walker = new AnimoAntlrWalker();
                return walker.expr(tree);
            }
            catch (AnimoException e)
            {
                throw new AnimoException("Include of file \"" +
                               file + "\" failed.");
            }
            catch (Exception e)
            {
                throw new AnimoException("Include of file \"" +
                               file + "\" failed: " + e);
            }
    }

    public void output(String file) {
      if (!hasLoadedBVH)
      {
          throw new AnimoException("This program never loads a BVH file!");
      }

      try
      {
          matrix.saveBVHfile(file);
      }
      catch (Exception e)
      {
          e.printStackTrace();
          throw new AnimoException("Failed to write file \"" +
                             file + "\": " + e);
      }
    }

    public void printGlobals() {
      System.out.println("GLOBAL FLOATING POINTS:");
      floats.printTable();
      System.out.println("\nFUNCTIONS:");
      funcs.printTable();
      System.out.println("\nBVH JOINTS:");
      joints.printTable();
    }

    private void error(String msg) {
      System.err.println(msg);
    }

}



//***************************************************************************
```

```
// AnimoDataType.java
//
// Represents a generic data type in ANIMO.  All objects stored in the symbol
// tables of an ANIMO program inherit from this class and are cast up to
// AnimoDataType before being stored in the tables and cast back down when the
// ANIMO programmer requests to perform an operation on that object.  If the
// operation is valid (e.g. calling add() on an AnimoFloatingPoint), the
// object's subclass defines that method, and it is called.  Otherwise (e.g.
// calling add() on an AnimoFunction), Java's rules will naturally call the
// corresponding method of AnimoDataType, which display the appropriate
// Invalid Operation error for that operation.
//
// Last Modified: 12/16/2005    By: Josh Poritz
//*****************************************************************************

import java.io.PrintWriter;

public class AnimoDataType
{
    public static String myType = "unknown";

    String name;   // Used in symbol table

    public AnimoDataType() {
        name = null;
    }

    public AnimoDataType(String name) {
        this.name = name;
    }

    public String typename() {
        return myType;
    }

    public AnimoDataType copy() {
        return new AnimoDataType();
    }

    public void setName(String name) {
        this.name = name;
    }

    // Display error: Wrong-type or uninitialized operand in unary operation
    public AnimoDataType badTypeError(String msg) {
      String error = this.typename() == "unknown" ?
          "Use of uninitialized variable " : "Illegal use of type <" +
          this.typename() + "> ";
      throw new AnimoException(error + "\"" +
                        (name != null ? name : "<?>") + "\"" +
                        " in unary " + msg + " operation");
    }

    // Display error: Wrong-type or unitialized operand in binary operation
    public AnimoDataType badTypeError(AnimoDataType b, String msg) {
      String error = this.typename() == "unknown" ?
          "Use of uninitialized variable " : "Illegal use of type <" +
```

```
      this.typename() + "> ";
  throw new AnimoException(error + "\"" +
                     (name != null ? name : "<?>") + "\" on " +
                     "left side of binary " + msg + " operation " +
                     "with " + (b.name != null ? "\"" + b.name +
                              "\"" : "constant " +
                      AnimoFloatingPoint.floatValue(b)));
}

// Display generic error
public AnimoDataType error(String msg) {
  throw new AnimoException("<" + typename() + "> " +
                     (name != null ? name : "<?>") + ": " + msg);
}

// Prints, if defined, name of this variable, to an output stream
public void print(PrintWriter w) {
    if (name != null)
        w.print(name + " = ");
    w.println("<undefined>");
}

// Prints the name specifically to stdout
public void print() {
    print(new PrintWriter(System.out, true));
}

// Prints the name AND type of this variable to an output stream
public void what(PrintWriter w) {
    w.print( "<" + typename() + "> ");
    print(w);
}

// Prints the name and type specifically to stdout
public void what() {
    what(new PrintWriter(System.out, true));
}

// ALL OTHER OPERATIONS, DEFINED BELOW, ARE ILLEGAL TO PERFORM ON THE
// GENERIC ANIMO DATA TYPE.

public AnimoDataType assign(AnimoDataType b) {
    return badTypeError(b, "=");
}

public AnimoDataType uminus() {
    return badTypeError("-");
}

public AnimoDataType plus(AnimoDataType b) {
    return badTypeError(b, "+");
}

public AnimoDataType add(AnimoDataType b) {
    return badTypeError(b, "+=");
}
```

```
public AnimoDataType minus(AnimoDataType b) {
    return badTypeError(b, "-");
}

public AnimoDataType sub(AnimoDataType b) {
    return badTypeError(b, "-=");
}

public AnimoDataType times(AnimoDataType b) {
    return badTypeError(b, "*");
}

public AnimoDataType mul(AnimoDataType b) {
    return badTypeError(b, "*=");
}

public AnimoDataType divby(AnimoDataType b) {
    return badTypeError(b, "/");
}

public AnimoDataType div(AnimoDataType b) {
    return badTypeError(b, "/=");
}

public AnimoDataType modulus(AnimoDataType b) {
    return badTypeError(b, "%");
}

public AnimoDataType rem(AnimoDataType b) {
    return badTypeError(b, "%=");
}

public AnimoDataType gt(AnimoDataType b) {
    return badTypeError(b, ">");
}

public AnimoDataType ge(AnimoDataType b) {
    return badTypeError(b, ">=");
}

public AnimoDataType lt(AnimoDataType b) {
    return badTypeError(b, "<");
}

public AnimoDataType le(AnimoDataType b) {
    return badTypeError(b, "<=");
}

public AnimoDataType eq(AnimoDataType b) {
    return badTypeError(b, "==");
}

public AnimoDataType ne(AnimoDataType b) {
    return badTypeError(b, "!=");
}

public AnimoDataType and(AnimoDataType b) {
```

```
        return badTypeError(b, "&&");
    }

    public AnimoDataType or(AnimoDataType b) {
        return badTypeError(b, "||");
    }

    public AnimoDataType not() {
        return badTypeError("!");
    }

    public AnimoDataType incr() {
      return badTypeError("++");
    }

    public AnimoDataType decr() {
      return badTypeError("--");
    }
}
```

```
//**************************************************************************
// AnimoException.java
//
// Represents an exception in ANIMO.  This class allows an ANIMO exception to
// be created and thrown (via inhering from RuntimeException).
//
// Last Modified: 11/06/2005    By: Josh Poritz
//**************************************************************************

public class AnimoException extends RuntimeException
{
    AnimoException(String msg) {
        System.err.println("Error: " + msg);
    }
}
```

```
//***************************************************************************
// AnimoFloatingPoint.java
//
// Represents a floating point variable in ANIMO.  This class defines all the
// legal operations that may be performed on a floating point operations and
// issues type checking on the other operand (e.g. ensures you don't add() an
// AnimoFunction to an AnimoFloatingPoint).
//
// Last Modified: 12/16/2005    By: Josh Poritz
//***************************************************************************

import java.io.PrintWriter;

public class AnimoFloatingPoint extends AnimoDataType
{
    static String myType = "float";

    double var;  // Value of the floating point variable

    public AnimoFloatingPoint(double x) {
        var = x;
    }

    public AnimoFloatingPoint(int x) {
      var = (double) x;
    }

    public AnimoFloatingPoint(boolean x) {
      var = x ? 1.0 : 0.0;
    }

    public String typename() {
        return myType;
    }

    public AnimoDataType copy() {
        return new AnimoFloatingPoint(var);
    }

    /*
     * Type-checking function called on the second operand to a binary
     * operator.  This operand must ALWAYS be an AnimoFloatingPoint.  If it
     * is, the operand is cast down to an AnimoFloatingPoint and returned.
     * Otherwise, compilation dies with an Invalid Cast message.
     */
    public static double floatValue(AnimoDataType b) {
        if (b instanceof AnimoFloatingPoint)
            return ((AnimoFloatingPoint)b).var;
        if (b instanceof AnimoBVHJoint)
          b.error("Illegal cast of " + ((AnimoBVHJoint)b).typename() +
                " to " + myType);
        if (b instanceof AnimoFunction)
          b.error("Illegal cast of " + ((AnimoFunction)b).typename() +
                " to " + myType);
      return Double.NaN;
    }
```

```
public double floatValue(AnimoDataType b, String msg)
{
  double cvrt = floatValue(b);
  if (!Double.isNaN(cvrt))
      return cvrt;

  String error = b.typename() == "unknown" ?
      "Use of uninitialized variable " : "Illegal use of type <" +
      b.typename() + "> ";
  throw new AnimoException(error + "\"" +
                     (b.name != null ? b.name : "<?>") + "\" on " +
                     "right side of binary " + msg +
                     " operation with \"" + this.name + "\"");
}

// Calls floatValue() on given parameter and converts the result to a
// boolean value.
public static boolean boolValue(AnimoDataType b) {
  return (floatValue(b) != 0.0);
}

public boolean boolValue(AnimoDataType b, String msg) {
  return (floatValue(b, msg) != 0.0);
}

// Writes name (if any) and value of this floating point to output stream
public void print(PrintWriter w) {
    if (name != null)
        w.print(name + " = ");
    w.println(Double.toString(var));
}

// ALL LEGAL FLOATING POINT OPERATIONS ARE DEFINED BELOW.

// -a
public AnimoDataType uminus() {
    return new AnimoFloatingPoint(-var);
}

// a + b
public AnimoDataType plus(AnimoDataType b) {
    return new AnimoFloatingPoint(var + floatValue(b, "+"));
}

// a += b
public AnimoDataType add(AnimoDataType b) {
    var += floatValue(b, "+=");
    return this;
}

// a - b
public AnimoDataType minus(AnimoDataType b) {
    return new AnimoFloatingPoint(var - floatValue(b, "-"));
}

// a -= b
public AnimoDataType sub(AnimoDataType b) {
```

```
    var -= floatValue(b, "-=");
    return this;
}

// a * b
public AnimoDataType times(AnimoDataType b) {
    return new AnimoFloatingPoint(var * floatValue(b, "*"));
}

// a *= b
public AnimoDataType mul(AnimoDataType b) {
    var *= floatValue(b, "*=");
    return this;
}

// a / b      Need to check for division by zero
public AnimoDataType divby(AnimoDataType b) {
  if (boolValue(b.eq(new AnimoFloatingPoint(0.0))))
      error("Division by zero");
  return new AnimoFloatingPoint(var / floatValue(b, "/"));
}

// a /= b     Need to check for division by zero
public AnimoDataType div(AnimoDataType b) {
  if (boolValue(b.eq(new AnimoFloatingPoint(0.0))))
      error("Division by zero");
  var /= floatValue(b, "/=");
    return this;
}

// a % b
public AnimoDataType modulus(AnimoDataType b) {
    return new AnimoFloatingPoint(var % floatValue(b, "%"));
}

// a %= b
public AnimoDataType rem(AnimoDataType b) {
    var %= floatValue(b, "%=");
    return this;
}

// a > b
public AnimoDataType gt(AnimoDataType b) {
    return new AnimoFloatingPoint(var > floatValue(b, ">"));
}

// a >= b
public AnimoDataType ge(AnimoDataType b) {
    return new AnimoFloatingPoint(var >= floatValue(b, ">="));
}

// a < b
public AnimoDataType lt(AnimoDataType b) {
    return new AnimoFloatingPoint(var < floatValue(b, "<"));
}

// a <= b
```

```
    public AnimoDataType le(AnimoDataType b) {
        return new AnimoFloatingPoint(var <= floatValue(b, "<="));
    }

    // a == b
    public AnimoDataType eq(AnimoDataType b) {
        return new AnimoFloatingPoint(var == floatValue(b, "=="));
    }

    // a != b
    public AnimoDataType ne(AnimoDataType b) {
        return new AnimoFloatingPoint(var != floatValue(b, "!="));
    }

    // a && b
    public AnimoDataType and(AnimoDataType b) {
        return new AnimoFloatingPoint((var != 0.0) && boolValue(b, "&&"));
    }

    // a || b
    public AnimoDataType or(AnimoDataType b) {
        return new AnimoFloatingPoint((var != 0.0) || boolValue(b, "||"));
    }

    // !a
    public AnimoDataType not() {
        return new AnimoFloatingPoint(var == 0.0);
    }

    // ++a
    public AnimoDataType incr() {
      ++var;
      return this;
    }

    // --a
    public AnimoDataType decr() {
      --var;
      return this;
    }
}
```

```
//*****************************************************************************
// AnimoFunction.java
//
// Represents any user-defined or internal functio in ANIMO.  A user-defined
// function is associated with an AST body, whereas an internal function is
// associated with a unique identifier, used to locate the function in the
// AnimoInternalFunction class.
//
// Last Modified: 11/06/2005    By: Josh Poritz
//*****************************************************************************

import java.io.PrintWriter;
import antlr.collections.AST;

public class AnimoFunction extends AnimoDataType
{
    static String myType = "function";

    // We need a reference to the AST for the function entry
    String[] args;
    AST body;              // body = null means an internal function.
    AnimoSymbolTable pst;  // the symbol table of static parent
    int id;                // for internal functions only

    // Constructor for new user-defined function
    public AnimoFunction(String name, String[] args,
                   AST body, AnimoSymbolTable pst) {
        super(name);
        this.args = args;
        this.body = body;
        this.pst = pst;
      this.id = -1;
    }

    // Constructor for new internal function
    public AnimoFunction(String name, int id) {
        super(name);
        this.args = null;
        this.id = id;
        pst = null;
        body = null;
    }

    public final boolean isInternal() {
        return body == null;
    }

    public final int getInternalId() {
        return id;
    }

    public String typename() {
        return myType;
    }

    public AnimoDataType copy() {
        return new AnimoFunction(name, args, body, pst);
```

```
    }

    // Writes name of this function and its parameter list to an output stream
    public void print(PrintWriter w) {
        if (body == null)
            w.println(name + " = <internal-function> #" + id);
        else
        {
            if (name != null)
                w.print(name + " = ");
            w.print( "<function>(" );
            for (int i = 0; ; i++)
            {
                w.print( args[i] );
                if (i >= args.length - 1)
                    break;
                w.print( "," );
            }
            w.println( ")" );
        }
    }

    // Write it specifically to stdout

    public String[] getArgs() {
        return args;
    }

    public AnimoSymbolTable getParentSymbolTable() {
        return pst;
    }

    public AST getBody() {
        return body;
    }
}
```

```
//***************************************************************************
// AnimoInternalFunction.java
//
// Represents an internal function in ANIMO.  This class defines ANIMO's
// built-in functions and built-in constants, a method for registering an
// internal function, and a method for executing an internal function.
//
// Last Modified: 12/17/2005    By: Josh Poritz
//***************************************************************************

public class AnimoInternalFunction
{
    static String null_data = "NULL";

    // Frames <-> Seconds conversion factors
    static double SECS_PER_FRAME = 1.0;
    static double FRAMES_PER_SEC = 1.0;

    // The latest time (excpressed in floating point seconds) that any
    // animation has been performed during thus far.
    private static double latestTime = 0.0;

    // The built-in functions and their internal identifiers
    final static int f_print = 0;
    final static int f_what = 1;
    final static int f_rotate = 2;
    final static int f_move = 3;
    final static int f_curtime = 4;
    final static int f_floor = 5;
    final static int f_ceil = 6;
    final static int f_sin = 7;
    final static int f_cos = 8;
    final static int f_tan = 9;
    final static int f_sqrt = 10;

    // Registers an internal function or constant with ANIMO
    public static void register(AnimoSymbolTable funcSt,
                        AnimoSymbolTable floatSt) {
      funcSt.put("print", new AnimoFunction(null, f_print));
      funcSt.put("what", new AnimoFunction(null, f_what));
      funcSt.put("rotate", new AnimoFunction(null, f_rotate));
      funcSt.put("move", new AnimoFunction(null, f_move));
      funcSt.put("curtime", new AnimoFunction(null, f_curtime));
      funcSt.put("floor", new AnimoFunction(null, f_floor));
      funcSt.put("ceil", new AnimoFunction(null, f_ceil));
      funcSt.put("sin", new AnimoFunction(null, f_sin));
      funcSt.put("cos", new AnimoFunction(null, f_cos));
      funcSt.put("tan", new AnimoFunction(null, f_tan));
      funcSt.put("sqrt", new AnimoFunction(null, f_sqrt));

        floatSt.put("E", new AnimoFloatingPoint(Math.E));
        floatSt.put("PI", new AnimoFloatingPoint(Math.PI));
    }

    // Executes an internal function
    public static AnimoDataType run(AnimoSymbolTable st, AnimoMatrix matrix,
                            int id, AnimoDataType[] params) {
```

```
AnimoBVHJoint joint;
double xRotPerFrame, yRotPerFrame, zRotPerFrame,
    xMovPerFrame, yMovPerFrame, zMovPerFrame;
double startTime, duration;
int curFrameNo;
double[] curRot, curPos;

  switch (id)
  {
    // print(...): Takes a list of AnimoDataType's, concatenates their
    // names together, writes them to stdout.
  case f_print:
    if (params == null)
      return new AnimoDataType(null_data);
      for (int i = 0; i < params.length; i++)
          params[i].print();
    return new AnimoDataType(null_data);

    // what(...): Takes a list of AnimoDataType's, concatenates their
    // name & type definitions together, writes them to stdout.
  case f_what:
    if (params == null)
    {
      st.printTable();
      return new AnimoDataType(null_data);
    }
    for (int i = 0; i < params.length; i++ )
      params[i].what();
    return new AnimoDataType(null_data);

    /*
     * rotate(AnimoBVHJoint joint, AnimoFloatingPoint startTime,
     *        AnimoFloatingPoint duration, AnimoFloatingPoint xRot,
     *        AnimoFloatingPoint yRot, AnimoFloatingPoint zRot):
     *  Sets a joint to be rotated (xRot, yRot, zRot) degrees starting
     *  at startTime for duration seconds.  The amount of rotation is
     *  distributed evenly over the frames from startTime to
     *  startTime + duration.
     */
  case f_rotate:
    // Parameter checking
      if (params == null || params.length != 6)
          throw new AnimoException("rotate() requires 6 parameters");
    if (!(params[0] instanceof AnimoBVHJoint))
      if (params[0].name != null)
          throw new AnimoException("rotate(): \"" + params[0].name +
                                   "\" is not a BVH joint.");
        else
          throw new AnimoException("rotate(): constant " +
                                   AnimoFloatingPoint.floatValue(params[0]) +
                                   " is not a BVH joint.");
    joint = (AnimoBVHJoint) params[0];
    for (int p = 1; p < params.length; p++)
      if (!(params[p] instanceof AnimoFloatingPoint))
          throw new AnimoException("rotate(): \"" + params[p].name +
                                   "\" is not a floating point.");
```

```
// Ensure valid startTime and duration
startTime = AnimoFloatingPoint.floatValue(params[1]);
duration = AnimoFloatingPoint.floatValue(params[2]);
if (startTime < 0)
  throw new
      AnimoException("rotate(): illegal negative start time: "
                 + startTime);
if (duration <= 0)
  throw new
      AnimoException("rotate(): illegal non-positive duration: "
                 + duration);

// Obtain secs <-> frames conversation factors
SECS_PER_FRAME = matrix.getFrameTime();
if (SECS_PER_FRAME <= 0)
  throw new AnimoException("Illegal non-positive frame rate " +
                     "specified in input SBVH file: " +
                     SECS_PER_FRAME);
FRAMES_PER_SEC = 1 / SECS_PER_FRAME;

// Calculate frame corresponding with startTime
curFrameNo = secToFrameRound(startTime);

// Get current rotation at that frame
curRot = matrix.getJointRotation(joint.jointName, curFrameNo - 1);

// Calculate amount of (x, y, z) rotation to perform per frame
xRotPerFrame =
   frameToSec((AnimoFloatingPoint.floatValue(params[3]) -
           curRot[0]) / duration);
yRotPerFrame =
   frameToSec((AnimoFloatingPoint.floatValue(params[4]) -
           curRot[1]) / duration);
zRotPerFrame =
   frameToSec((AnimoFloatingPoint.floatValue(params[5]) -
           curRot[2]) / duration);
// Calculate frame corresponding with startTime + duration, and
// distrubte the amount of rotation evenly over those frames.
for ( ;
   curFrameNo < secToFrameRound(startTime + duration);
   ++curFrameNo)
{
  curRot[0] += xRotPerFrame;
  curRot[1] += yRotPerFrame;
  curRot[2] += zRotPerFrame;

  matrix.setJointRotation(joint.jointName, curFrameNo, curRot);
}

// Re-calculate latestTime
latestTime = max(latestTime, startTime + duration);

// No return value
return new AnimoDataType(null_data);

/*
```

```
   * move(AnimoFloatingPoint startTime, AnimoFloatingPoint duration,
   *      AnimoFloatingPoint xMov, AnimoFloatingPoint yMov,
   *      AnimoFloatingPoint zMov):
   *  Sets the root joint to be moved (xMov, yMov, zMov) units
   *  starting at startTime for duration seconds.  The amount of
   *  movement is distributed evenly over the frames from startTime
   *  to startTime + duration.
   */
case f_move:
  // Parameter checking
    if (params == null || params.length != 5)
        throw new AnimoException("move() requires 5 parameters");
  for (int p = 0; p < params.length; p++)
    if (!(params[p] instanceof AnimoFloatingPoint))
        throw new AnimoException("move(): \"" + params[p].name +
                           "\" is not a floating point.");

  // Ensure valid startTime and duration
  startTime = AnimoFloatingPoint.floatValue(params[0]);
  duration = AnimoFloatingPoint.floatValue(params[1]);
  if (startTime < 0)
    throw new
        AnimoException("move(): illegal negative start time: " +
                   startTime);
  if (duration <= 0)
    throw new
        AnimoException("move(): illegal non-positive duration: " +
                   duration);

  // Obtain secs <-> frames conversation factors
  SECS_PER_FRAME = matrix.getFrameTime();
  if (SECS_PER_FRAME <= 0)
    throw new AnimoException("Illegal non-positive frame rate " +
                       "specified in input SBVH file: " +
                       SECS_PER_FRAME);
  FRAMES_PER_SEC = 1 / SECS_PER_FRAME;

  // Calculate frame corresponding with startTime
  curFrameNo = secToFrameRound(startTime);

  // Get current position at that frame
  curPos = matrix.getRootPos(curFrameNo - 1);

  // Calculate amount of (x, y, z) movement to perform per frame
  xMovPerFrame =
    frameToSec((AnimoFloatingPoint.floatValue(params[2]) -
              curPos[0]) / duration);
  yMovPerFrame =
    frameToSec((AnimoFloatingPoint.floatValue(params[3]) -
              curPos[1]) / duration);
  zMovPerFrame =
    frameToSec((AnimoFloatingPoint.floatValue(params[4]) -
              curPos[2]) / duration);

  // Calculate frame corresponding with startTime + duration, and
  // distrubte the amount of movement evenly over those frames.
  for ( ;
```

```
       curFrameNo < secToFrameRound(startTime + duration);
       ++curFrameNo)
    {
      curPos[0] += xMovPerFrame;
      curPos[1] += yMovPerFrame;
      curPos[2] += zMovPerFrame;

      matrix.setRootPos(curFrameNo, curPos);
    }

    // Re-calculate latestTime
    latestTime = max(latestTime, startTime + duration);

    // No return value
    return new AnimoDataType(null_data);

    // curtime(): Returns the latest time at which any animation has
    // been performed thus far.
case f_curtime:
    if (params != null)
      throw new AnimoException("curtime() should be passed no " +
                        "parameters.");
    return new AnimoFloatingPoint(latestTime);

    // floor(x): Return largest integer that is <= x
case f_floor:
    if (params == null || params.length != 1)
      throw new AnimoException("floor() requires exactly one " +
                        "parameter.");
    if (!(params[0] instanceof AnimoFloatingPoint))
      throw new AnimoException("floor(): argument " + params[0].name
                        + " is not a floating point.");
    return new AnimoFloatingPoint(
        Math.floor(AnimoFloatingPoint.floatValue(params[0])));

    // ceil(x): Return smallest integer that is >= x
case f_ceil:
    if (params == null || params.length != 1)
      throw new AnimoException("ceil() requires exactly one " +
                        "parameter.");
    if (!(params[0] instanceof AnimoFloatingPoint))
      throw new AnimoException("ceil(): argument " + params[0].name
                        + " is not a floating point.");
    return new AnimoFloatingPoint(
        Math.ceil(AnimoFloatingPoint.floatValue(params[0])));

    // sin(x): Return the trigonometric sine of x
case f_sin:
    if (params == null || params.length != 1)
      throw new AnimoException("sin() requires exactly one " +
                        "parameter.");
    if (!(params[0] instanceof AnimoFloatingPoint))
      throw new AnimoException("sin(): argument " + params[0].name
                        + " is not a floating point.");
    return new AnimoFloatingPoint(
        Math.sin(AnimoFloatingPoint.floatValue(params[0])));
```

```
      // cos(x): Return the trigonometric cosine of x
  case f_cos:
      if (params == null || params.length != 1)
        throw new AnimoException("cos() requires exactly one " +
                          "parameter.");
      if (!(params[0] instanceof AnimoFloatingPoint))
        throw new AnimoException("cos(): argument " + params[0].name
                          + " is not a floating point.");
      return new AnimoFloatingPoint(
          Math.cos(AnimoFloatingPoint.floatValue(params[0])));

      // tan(x): Return the trigonometric tangent of x
  case f_tan:
      if (params == null || params.length != 1)
        throw new AnimoException("tan() requires exactly one " +
                          "parameter.");
      if (!(params[0] instanceof AnimoFloatingPoint))
        throw new AnimoException("tan(): argument " + params[0].name
                          + " is not a floating point.");
      return new AnimoFloatingPoint(
          Math.tan(AnimoFloatingPoint.floatValue(params[0])));

      // sqrt(x): Return the trigonometric sqrte of x
  case f_sqrt:
      if (params == null || params.length != 1)
        throw new AnimoException("sqrt() requires exactly one " +
                          "parameter.");
      if (!(params[0] instanceof AnimoFloatingPoint))
        throw new AnimoException("sqrt(): argument " + params[0].name
                          + " is not a floating point.");
      return new AnimoFloatingPoint(
          Math.sqrt(AnimoFloatingPoint.floatValue(params[0])));

      // Unkown internal function
    default:
        throw new AnimoException("Unknown internal function");
    }
}

// frame --> seconds conversion
public static double frameToSec(double frame) {
  return frame * SECS_PER_FRAME;
}

// seconds --> frame conversion
public static double secToFrame(double sec) {
  return sec * FRAMES_PER_SEC;
}

// convert seconds --> frame and rounds the result to an exact frame #
public static int secToFrameRound(double sec) {
  return (int)Math.round(secToFrame(sec));
}

// Returns the larget of two integers
public static double max(double a, double b) {
  return a > b ? a : b;
```

```
    }
}
```

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

/**
 * The main class
 *
 * @@author Joshua Poritz - jsp2104@@columbia.edu
 */
class AnimoMain {
    static String outputFile = "out.bvh";
    static boolean verbose = false;

    public static void execFile( String filename ) {
        try
        {
            InputStream input = ( null != filename ) ?
                (InputStream) new FileInputStream( filename ) :
                (InputStream) System.in;

            AnimoAntlrLexer lexer = new AnimoAntlrLexer( input );

            AnimoAntlrParser parser = new AnimoAntlrParser( lexer );

            // Parse the input program
            parser.program();

            if ( lexer.nr_error > 0 || parser.nr_error > 0 )
            {
                System.err.println( "Parsing errors found. Stop." );
                return;
            }

            CommonAST tree = (CommonAST)parser.getAST();

            if ( verbose )
            {
                // Print the resulting tree out in LISP notation
                System.out.println(
                    "================ tree structure ======================" );
                System.out.println( tree.toStringList() );
            }

            AnimoAntlrWalker walker = new AnimoAntlrWalker();

            if ( verbose )
                System.out.println(
                    "================ program output ======================" );

          try
          {
            // Traverse the tree created by the parser
            AnimoDataType r = walker.expr( tree );
```

```
        // Write the BVH output file
        walker.cmpl.output(outputFile);
      }
      catch (AnimoException e)
      {
        System.out.println("Compilation aborted due to compile-time " +
                       "errors.");
        if ( verbose )
        {
            System.out.println(
            "================ stack trace ========================" );
            e.printStackTrace();
        }
      }
      catch (Exception e)
      {
        System.out.println("Compilation aborted due to unexpected " +
                       "error in ANIMO compiler: " + e);
        if ( verbose )
        {
            System.out.println(
            "================ stack trace ========================" );
            e.printStackTrace();
        }
      }

        if ( verbose )
        {
            System.out.println(
                "================ global variables ====================" );
            walker.cmpl.printGlobals();
        }

    } catch( IOException e ) {
        System.err.println( "Error: I/O: " + e );
    } catch( RecognitionException e ) {
        System.err.println( "Error: Recognition: " + e );
    } catch( TokenStreamException e ) {
        System.err.println( "Error: Token stream: " + e );
    } catch( Exception e ) {
        System.err.println( "Error: " + e );
    }
}

public static void main( String[] args ) {

  String inputFile = null;
  boolean skip = false;

  for (int arg = 0; arg < args.length; arg++)
  {
      if (skip)
      {
        skip = false;
        continue;
      }
```

```
            if (args[arg].equals("-v"))
              verbose = true;
            else if (args[arg].equals("-o"))
            {
              if (args.length < arg + 1)
              {
                  System.err.println("-o specified with no output file");
                  System.exit(255);
              }

              outputFile = args[arg + 1];
              skip = true;
            }
            else
            {
              if (inputFile == null)
                  inputFile = args[arg];
                else
              {
                  System.err.println("Multiple input files not allowed");
                  System.exit(255);
              }
            }
        }

        execFile(inputFile);
          System.exit( 0 );
      }
}
```

```
/************************************************
 * AnimoMain.java       Author: Alexei Masterov
 ************************************************/

import java.io.*;
import java.util.*;
import bvh.*;
import bvh.parser.*;

public class AnimoMatrix
{
      private BVHFile bvhFile;
      private JointCache myJointCache;

      // load a bvh file from disk
      // parameter is a filename with path
    public
    int loadBVHfile( String fileName ) throws IOException, ParseException
    {
            bvhFile = BVHParser.parseFile( new File(fileName) );
            myJointCache = new JointCache( bvhFile.getRoot() );
            myJointCache.refresh();
            return 0;
    }

    /* * */

      // save a bvh file to disk
      // parameter is a filename with path
    public
    int saveBVHfile ( String fileName ) throws IOException
    {
            BVHGenerator myBVHGenerator = new BVHGenerator (bvhFile, new
File(fileName) );
            myBVHGenerator.generate( false );
            return 0;
    }

    /* * */

      // returns 3d array - the current rotation for the joint named "jointName"
    public
    double[] getJointRotation( String jointName, int frameNumber )
    {
            double [] jointRotation = new double[3];

            Joint myJoint = myJointCache.getJoint(jointName);
            if ( myJoint == null )
                  throw new AnimoException("getJointRotation: unknown joint: " +
jointName);

            // if Frame number is < 0 - return joint's initial position
            if ( frameNumber < 0 )
            {
                  jointRotation[0] = myJoint.getX();
                  jointRotation[1] = myJoint.getY();
                  jointRotation[2] = myJoint.getZ();
```

```
            }
            // if it is out of range - return all zeros
            else if ( frameNumber > bvhFile.getMotion().getFrameCount()-1 )
            {
                  jointRotation[0] = 0;
                  jointRotation[1] = 0;
                  jointRotation[2] = 0;
            }
            else // get the values from the array
            {
                  double [] myFrame = bvhFile.getMotion().getFrame(frameNumber);
                  // calculate the offset
                  int firstChannel = myJoint.getFirstChannel() +
(myJoint.getNumChannels() - 3);

                  jointRotation[0] = myFrame[firstChannel];
                  jointRotation[1] = myFrame[firstChannel+1];
                  jointRotation[2] = myFrame[firstChannel+2];
            }

            return jointRotation;
    }

    /* * */

    public
    void setJointRotation( String jointName,
                        int frameNumber, double [] jointRotation )
      throws AnimoException
    {
            if ( jointRotation.length != 3 )
                  throw new AnimoException("setJointRotation: array size must be
3");

            Joint myJoint = myJointCache.getJoint(jointName);
            if ( myJoint == null )
                  throw new AnimoException("setJointRotation: unknown joint: " +
jointName);

            // calculate the offset
            int firstChannel = myJoint.getFirstChannel() +
(myJoint.getNumChannels() - 3);

            bvhFile.getMotion().setData(frameNumber, firstChannel  ,
jointRotation[0]);
            bvhFile.getMotion().setData(frameNumber, firstChannel+1,
jointRotation[1]);
            bvhFile.getMotion().setData(frameNumber, firstChannel+2,
jointRotation[2]);
    }

    /* * */

    public
    double[] getRootPos( int frameNumber )
    {
            double [] position = new double[3];
```

```
            Joint myJoint = bvhFile.getRoot();
            if ( myJoint == null )
                  throw new AnimoException("getRootPos: Cannot getRoot");

            // if it is out of range - return all zeros
            if ( frameNumber < 0 || frameNumber >
                 bvhFile.getMotion().getFrameCount()-1 )
            {
                  position[0] = 0;
                  position[1] = 0;
                  position[2] = 0;
            }
            else // get the values from the array
            {
                  double [] myFrame = bvhFile.getMotion().getFrame(frameNumber);
                  // calculate the offset
                  int firstChannel = myJoint.getFirstChannel();

                  position[0] = myFrame[firstChannel];
                  position[1] = myFrame[firstChannel+1];
                  position[2] = myFrame[firstChannel+2];
            }

            return position;
    }

    /* * */

    public
    void setRootPos( int frameNumber, double [] position)
            throws AnimoException
    {
            if ( position.length != 3 )
                  throw new AnimoException("setRootPos: array size must be 3");

            Joint myJoint = bvhFile.getRoot();
            if ( myJoint == null )
                  throw new AnimoException("setRootPos: Cannot getRoot");

            // calculate the offset
            int firstChannel = myJoint.getFirstChannel();

            bvhFile.getMotion().setData(frameNumber, firstChannel  ,
position[0]);
            bvhFile.getMotion().setData(frameNumber, firstChannel+1,
position[1]);
            bvhFile.getMotion().setData(frameNumber, firstChannel+2,
position[2]);
    }

    /* * */

      public
      Joint getJoint(String name)
      {
            return myJointCache.getJoint(name);
      }
```

```
      public
      Set getAllJoints()
      {
            return myJointCache.getAllJoints();
      }

      public
      void listAllJoints()
      {
            Set allJoints = getAllJoints();

            // Iterating over the elements in the set
            Iterator it = allJoints.iterator();
            while (it.hasNext()) {
                  System.out.println( it.next() );
            }
      }

    /* * */

      public
      int getChannelCount()
      {
            return bvhFile.getMotion().getChannelCount();
      }

    /* * */

      public
      int getFrameCount()
      {
            return bvhFile.getMotion().getFrameCount();
      }

    /* * */

      public
      double getFrameTime()
      {
            return bvhFile.getMotion().getFrameTime();
      }

    /* * */

      // local main for module testing
      public static void main ( String args[] ) throws Exception
      {
            AnimoMatrix myAnimoMatrix = new AnimoMatrix();

            System.out.println("Loading \"" + args[0] + "\"" );
            myAnimoMatrix.loadBVHfile( args[0] );

            System.out.println("Number of Channels: " +
myAnimoMatrix.getChannelCount() );
            System.out.println("Number of Frames: " +
myAnimoMatrix.getFrameCount() );
```

```
            System.out.println("Joints:");
            myAnimoMatrix.listAllJoints();

            System.out.println("Getting rotation for Head" );
            double [] myRot = myAnimoMatrix.getJointRotation( "Head", 10 );
            System.out.println("Head, frame 10 :" + myRot[0] + ", " + myRot[1] +
", " + myRot[2]);

            System.out.println("Getting Root Position" );
            myRot = myAnimoMatrix.getRootPos( 10 );
            System.out.println("RootPos, frame 10 :" + myRot[0] + ", " +
myRot[1] + ", " + myRot[2]);


            myRot[0] = 1.1;
            myRot[1] = 2.2;
            myRot[2] = 3.3;

            System.out.println("Setting rotation for Hips" );
            myAnimoMatrix.setJointRotation( "Hips", 12, myRot );

            System.out.println("Setting Root Position" );
            myAnimoMatrix.setRootPos( 12, myRot );

            System.out.println("Saving \"./mytest.bvh\"" );
            myAnimoMatrix.saveBVHfile("./mytest.bvh");

            System.out.println("AnimoMatrix done!");
      }
}
```

```
/*******************************************
 * AnimoSymbolTable.java
 *
 * Uses a hashtable to store keys and values.
 *
 * Author: Natasha Shamis
 *******************************************/

import java.util.*;
import java.io.*;

public class AnimoSymbolTable
{
    private Hashtable table;
    private AnimoSymbolTable parent;
    private AnimoSymbolTable current;
    private String name;

    //Constructor for parent scope
    public AnimoSymbolTable(String name) {
      table = new Hashtable();
      this.parent = null;
      this.name = name;
    }

    //Constructor for child scope
    public AnimoSymbolTable(String name, AnimoSymbolTable parent) {
      table = new Hashtable();
      this.parent = parent;
      this.name = name;
    }

    //Get parent scope
    public AnimoSymbolTable getParent() {
      return parent;
    }

    //Get table name
    public String getName() {
      return name;
    }

    //Print all keys and corresponding values in table
    public void printTable() {
        System.out.println(table.toString());
    }

    //Given a key, return corresponding value
    public AnimoDataType get(String key) {
      AnimoSymbolTable current = this;
      while (current != null)
          if (current.table.containsKey(key))
              return (AnimoDataType) current.table.get(key);
          else
            current = current.parent;//if key not found, search parent scope
      return null;
    }
```

```
    //Add table entry
    public void put(String key, AnimoDataType value) {
      table.put(key, value);
    }

}//AnimoSymbolTable
```

```
/*************************************
 * tBuiltIns.animo
 * Test program for built-in functions
 * Author: Josh Poritz
 *************************************/

include "test.sbvh";

a = 3;
print(3);
print(a);
print(a, 5, a + 1);
print(Neck, a);
print();
// print(what(3));

what(3);
what(a);
what(a, 5, a + 1);
what(Neck, ++a);
what();

start = curtime();
rotate(Neck, 0, 4, 15, 30, 45);
end = curtime();
duration = end - start;
print(duration);
// curtime(3);

f1 = floor(3);
f2 = floor(3.98);
f3 = floor(-0.05);
print(f1, f2, f3);
// f4 = floor();
// f5 = floor(0, 2);
f6 = floor(a);
print(f6);
// f7 = floor(bluh);
// f8 = floor(Neck);

c1 = ceil(3);
c2 = ceil(3.98);
c3 = ceil(-0.05);
print(c1, c2, c3);
// c4 = ceil();
// c5 = ceil(0, 2);
c6 = ceil(a);
print(c6);
// c7 = ceil(bluh);
// c8 = ceil(Neck);

co1 = cos(3);
co2 = cos(3.98);
co3 = cos(-0.05);
print(co1, co2, co3);
// co4 = cos();
// co5 = cos(0, 2);
```

```
co6 = cos(a);
print(co6);
// co7 = cos(bluh);
// co8 = cos(Neck);

si1 = sin(3);
si2 = sin(3.98);
si3 = sin(-0.05);
print(si1, si2, si3);
// si4 = sin();
// si5 = sin(0, 2);
si6 = sin(a);
print(si6);
// si7 = sin(bluh);
// si8 = sin(Neck);

ta1 = tan(3);
ta2 = tan(3.98);
ta3 = tan(-0.05);
print(ta1, ta2, ta3);
// ta4 = tan();
// ta5 = tan(0, 2);
ta6 = tan(a);
print(ta6);
// ta7 = tan(bluh);
// ta8 = tan(Neck);

sq1 = sqrt(3);
sq2 = sqrt(3.98);
sq3 = sqrt(-0.05);
print(sq1, sq2, sq3);
// sq4 = sqrt();
// sq5 = sqrt(0, 2);
sq6 = sqrt(a);
print(sq6);
// sq7 = sqrt(bluh);
// sq8 = sqrt(Neck);

print(PI);
print(E);
```

```
/****************
 * test.sbvh
 ***************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 0
Frame Time: 1.0000
```

```
/*****************************
 * tFor.animo
 * Test program for loops
 * Author: Joshua Poritz
 *****************************/

include "test.sbvh";

// simple for loop only involving constants
for (i = 1 to 3)
    if (i == 1)
      rotate(Neck, 0, 1, 3, 3, 3);
    else if (i == 2)
      rotate(Neck, 1, 1, 4, 4, 4);
    else
      rotate(Neck, 2, 1, 5, 5, 5);


// using the loop constant inside the loop
for (j = 10 to 15)
    rotate(LeftCollar, j - 10, 1, j, j, j);

// using variables for the loop bounds
k = 20;
l = 25;
for (i = k to l)
    rotate(LeftAnkle, i, 1, i, i, i);

// using expressions for the loop bounds
m = 3;
n = -350;
for (i = m * 10 to (-n / 10) % 100)
    rotate(RightAnkle, i, 1, i, i, i);

// lower bound is a negative number
for (i = -4 to 4)
    rotate(Hips, i + 44, 1, i + 44, i + 44, i + 44);

// lower and upper bounds are negative numbers
for (i = -10 to -5)
    rotate(LeftKnee, 60 + i, 1, 60 + i, 60 + i, 60 + i);

// upper bound = lower bound
for (i = 58 to 58)
    rotate(LeftShoulder, i, 1, i, i, i);

// upper bound < lower bound
for (i = 60 to 59)
    rotate(LeftElbow, i, 1, i, i, i);

// multiline for loop
for (j = 70 to 71)
{
    if (j == 70)
      rotate(Head, 70, 1, 70, j, 70);
    if (j == 71)
      rotate(Head, 71, 1, 71, j, 71);
```

```
}

// nested multiline for loop
for (k = 1 to 4)
{
    rotate(Head, 10 + k, 1, 72 + k, 72 + k, 72 + k);

    for (n = 3 to 4)
      rotate(RightAnkle, (n + 5) * 10 + k, 1,
             (n + 5) * 10 + k, (n + 5) * 10 + k, (n + 5) * 10 + k);

    rotate(Head, 97, 1, 97, 97, 97);
}

// Illegal use of uninitialized variables as bounds
/*
for (i = 5 to duh)
    rotate(Head, 99, 1, 99, 99, 99);
for (i = duh to 5)
    rotate(Head, 100, 1, 100, 100, 100);
*/

// Illegal use of BVH joints as bounds
/*
for (i = 5 to Neck)
    rotate(Head, 99, 1, 99, 99, 99);
for (i = Neck to 5)
    rotate(Head, 100, 1, 100, 100, 100);
*/

// Illegal use of BVH joint as loop counter
/*
for (Neck = 1 to 5)
    rotate(Head, 101, 1, 101, 101, 101);
*/
```

```
/*********************
 * tFor.bvh
 ***********************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 98
Frame Time: 1.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 10.0000 10.0000 10.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000 3.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 11.0000 11.0000 11.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 4.0000 4.0000 4.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 12.0000 12.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 13.0000 13.0000 13.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 14.0000 14.0000 14.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 73.0000
73.0000 73.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 74.0000
74.0000 74.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 75.0000
75.0000 75.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 20.0000 20.0000 20.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 21.0000 21.0000 21.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 22.0000 22.0000 22.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 23.0000 23.0000 23.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 24.0000 24.0000 24.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 76.0000
76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 31.0000
31.0000 31.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 32.0000
32.0000 32.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 33.0000
33.0000 33.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 34.0000
34.0000 34.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 40.0000 40.0000 40.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 41.0000 41.0000 41.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 42.0000 42.0000 42.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 43.0000 43.0000 43.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 44.0000 44.0000 44.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 45.0000 45.0000 45.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 46.0000 46.0000 46.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 47.0000 47.0000 47.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 35.0000
35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000
76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 50.0000
50.0000 50.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 51.0000
51.0000 51.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 52.0000
52.0000 52.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 53.0000
53.0000 53.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 54.0000
54.0000 54.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000

0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000

58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 76.0000 76.0000 76.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 70.0000 70.0000 70.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000

0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 35.0000 35.0000 35.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 81.0000 81.0000 81.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 82.0000 82.0000 82.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 83.0000 83.0000 83.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000

58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 84.0000 84.0000 84.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 91.0000 91.0000 91.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 92.0000 92.0000 92.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 93.0000 93.0000 93.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000

0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 94.0000 94.0000 94.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 94.0000 94.0000 94.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 94.0000 94.0000 94.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 71.0000 71.0000 71.0000
0.0000 0.0000 0.0000 48.0000 48.0000 48.0000 0.0000 0.0000 0.0000 55.0000
55.0000 55.0000 25.0000 25.0000 25.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 94.0000 94.0000 94.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000
58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000
5.0000 5.0000 97.0000 97.0000 97.0000

```
/****************************************
 * tFunc.animo
 * Test for functions
 * Author: Joshua Poritz
 ****************************************/

include "test.sbvh";

// function call: no parameters, no return value
func one()
{
    rotate(LeftKnee, 0, 1, 3, 3, 3);
}
one();

// function call: one paramter, no return value
func two(x)
{
    rotate(LeftElbow, x, 1, x, x, x);
}
two(5);
two(6.2);

// function call: multiple parameters, no return value
func three(x, a, b, c)
{
    rotate(LeftAnkle, x, 1, a, b, c);
}
three(10, 10.1, 10.2, 10.3);
three(11, 11.1, 11.2, 11.3);

// function call: passing a variable to a function
func four(x)
{
    rotate(Neck, 12, 1, x, x, x);
}
rotAmt = 12.5;
four(rotAmt);

// function call: multiple parameters, floating point return value
func five(m, n)
{
    move(m, 1, n, n, n);
    return 3.98;
}
p = five(13, 14);
rotate(LeftWrist, 15, 1, p, p, p);

// function call: manipulating a parameter and returning it to caller
func six(t, u)
{
    rotate(Head, t, 1, u, u, u);
    ++u;
    return u;
}
q = six(20, 21);
rotate(RightWrist, 22, 1, q, q, q);
```

```
// function call: using a return value where there is none
/* 12/15/2005: THIS IS NOW DISALLOWED
func seven(foo)
{
    rotate(LeftElbow, 23, 1, 24, 24, foo);
}
r = seven(25);
rotate(RightElbow, 26, 1, r + 27, r + 27, 27 + r);
*/

// function call: using a return value where there is none #2
/* 12/15/2005: THIS IS NOW DISALLOWED
func _seven(foo)
{
    rotate(LeftElbow, 23, 24, 24, foo);
    return;
}
r = _seven(25);
rotate(RightElbow, 26, 1, r + 27, r + 27, 27 + r);
*/

// function call: discarding the return value (and parameter)
func eight(bar)
{
    rotate(RightAnkle, 28, 1, 28, 28, 28);
    return 1.9;
}
eight(2.975);

// function call: mismatching number of parameters
/*
func nine(bar)
{
    rotate(RightAnkle, 30, 1, 7.1, 7.1, 7.1);
}
nine();
nine(3, 4);
*/

// function call: mismatching parameter types
/*
func ten(a, b)
{
    rotate(a, 30.5, 1, b, 8.1, 8.1);
}
ten(5, 8.1);
ten(Neck, Neck);
*/

// function call: calling the function before it's declared
/*
eleven(9.1, 9.1);
func eleven(x, y)
{
    rotate(Neck, 31.5, x, y, y, y);
}
```

```
*/

// nested functions (should be disallowed by grammar)
/*
func twelve()
{
    rotate(Neck, 32.5, 10.1, 10.2, 10.3);
    func thirteen()
    {
      rotate(Neck, 33.5, 10.4, 10.5, 10.6);
    }
    thirteen();
}
twelve();
*/

// calling a function and passing its return value as param to other function
// in the midst of a complex expression
func fourteen(a, coolParam) {
    rotate(Hips, 50, 1, a, coolParam, coolParam);
}
func fifteen(coolerParam) {
    move(50, 1, coolerParam, coolerParam, coolerParam);
    return coolerParam + 1;
}
fourteen(50, 2 * fifteen(17.2) + 14.4);

// function call: more interesting multi-line function
func sixteen(start) {
    for (i = start to start + 5)
      if (i % 2 == 0)
          rotate(LeftAnkle, i, 1, i, i, i);
      else
          rotate(RightAnkle, i, 1, i, i, i);
}
sixteen(55);

// Illegal function prototype with constant (should be parse error)
/*
func seventeen(3) {
}
*/

// Illegal function prototype with expression (should be parse error)
/*
func eighteen(a + 5) {
}
*/

// Illegal function prototype with BVH joint
/*
func nineteen(Neck) {
    print(Neck);
}
nineteen(Neck);
*/
```

```
/****************
 * tFunc.bvh
 ****************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 61
Frame Time: 1.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
5.0000 5.0000 5.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 10.1000 10.2000 10.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000

14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 0.0000
0.0000 0.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 21.0000
21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.5000 12.5000 12.5000 21.0000
21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000

14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
14.0000 14.0000 14.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 3.0000
3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.0000
28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000 12.5000
21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
28.0000 28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
28.0000 28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
28.0000 28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
28.0000 28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
28.0000 28.0000 28.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 11.1000 11.2000 11.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
55.0000 55.0000 55.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000

17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 56.0000 56.0000 56.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
55.0000 55.0000 55.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 56.0000 56.0000 56.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
57.0000 57.0000 57.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
57.0000 57.0000 57.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 58.0000 58.0000 58.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
59.0000 59.0000 59.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000
17.2000 17.2000 17.2000 50.0000 50.8000 50.8000 0.0000 0.0000 0.0000 3.0000
3.0000 3.0000 60.0000 60.0000 60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
59.0000 59.0000 59.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 6.2000 6.2000 6.2000 3.9800 3.9800 3.9800 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 22.0000 22.0000 22.0000 12.5000 12.5000
12.5000 21.0000 21.0000 21.0000

```
/*************************************
 * tIf.animo
 * Test for conditionals
 * Author: Joshua Poritz
 *************************************/

include "test.sbvh";

x = 5;

// regular if with equality
if (x == 5)
    rotate(LeftKnee, 0, 1, 15, 15, 15);         // exec

// equality if with else clause
if (x == 3)
    rotate(LeftKnee, 1, 1, 30, 30, 30);
else
    rotate(LeftKnee, 1, 1, 45, 45, 45);         // exec

// equality "else if"
if (x == 1)
    move(10, 1, 100, 100, 100);
else if (x == 5)
    move(11, 1, 105, 105, 105);                 // exec

// testing a variable for truth
if (x)
    move(12, 1, 110, 110, 110);                 // exec

// if with && operator
if (x == 3 && 0)
    rotate(Chest, 2, 1, 6.2, 6.2, 6.2);
if (x == 3 && 1)
    rotate(Neck, 2, 1, 2.1, 2.1, 2.1);
if (x == 5 && 0)
    rotate(LeftKnee, 2, 1, 60, 60, 60);
if (x == 5 && 1)
    rotate(RightKnee, 2, 1, 20, 20, 20);  // exec

// if with || operator
if (x == 3 || 0)
    rotate(Chest, 3, 1, 80, 80, 80);
if (x == 3 || 1)
    rotate(Neck, 3, 1, 81, 81, 81);       // exec
if (x == 5 || 0)
    rotate(LeftKnee, 3, 1, 82, 82, 82);         // exec
if (x == 5 || 1)
    rotate(RightKnee, 3, 1, 83, 83, 83);  // exec

// if statement with negation
if (!(x))
    rotate(RightKnee, 4, 1, 83.5, 83.5, 83.5);

// if statement with negation and operator
if (!(x * 0))
    rotate(LeftKnee, 4, 1, 85, 85, 85);         // exec
```

```
// if statement with increment
y = x;
if (++y)
    rotate(Neck, 7, 1, 83.7, 83.7, 83.7);       // exec

// if statement with negative number
if (-5)
    rotate(Neck, 8, 1, 83.8, 83.8, 83.8);       // exec

// if with complex condition
if (!((x + 3) % 4) && (x + 1 > 5 || x * 0))
    rotate(LeftKnee, 5, 1, 86, 86, 86);         // exec

// Multi-statement if
if (x == 5)
{
    rotate(LeftKnee, 6, 1, 87, 87, 87);         // exec
    move(6, 1, 17, 17, 17);               // exec
}

// Nested "if"s
if (x == 5)
    if (x + 1 == 6)
      rotate(LeftKnee, 7, 1, 88, 88, 88); // exec
    else
      move(7, 1, 21, 21, 21);
else
    rotate(Neck, 7, 1, 90, 90, 90);

// Nested multi-statement "if"s with else clauses
if (x == 5)
{
    rotate(LeftAnkle, 8, 1, 94, 94, 94);  // exec

    if (x + 1 != 6)
      rotate(LeftKnee, 8, 1, 93, 93, 93);
    else
    {
      move(8, 1, 96, 96, 96);             // exec
      move(9, 1, 97, 97, 97);             // exec
    }

    rotate(LeftAnkle, 9, 1, 95, 95, 95);  // exec
}
else
    rotate(Neck, 8, 1, 96, 96, 98);

/*
if (bluh)   // Illegal testing of uninitialized variable.
    q = 8;
if (Neck)   // Illegal testing of non floating point variable.
    q = 2;
*/


/*******************************
```

```
 * tIf.bvh
 ****************************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
        OFFSET 8.0000 0.0000 0.0000
```

```
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftElbow
        {
          OFFSET 0.0000 -12.0000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT LeftWrist
          {
            OFFSET 0.0000 -9.5000 0.0000
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET 0.0000 -7.0000 0.0000
            }
          }
        }
      }
    }
    JOINT RightCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightShoulder
      {
        OFFSET -8.0000 0.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightElbow
        {
          OFFSET 0.0000 -12.0000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT RightWrist
          {
            OFFSET 0.0000 -9.5000 0.0000
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET 0.0000 -7.0000 0.0000
            }
          }
        }
      }
    }
    JOINT Neck
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT Head
      {
        OFFSET 0.0000 6.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 7.0000 0.0000
        }
      }
    }
  }
}
```

MOTION
Frames: 13
Frame Time: 1.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000
15.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 45.0000 45.0000
45.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 45.0000 45.0000
45.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 20.0000 20.0000 20.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 82.0000 82.0000
82.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.0000 83.0000 83.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 81.0000 81.0000 81.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 85.0000 85.0000
85.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.0000 83.0000 83.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 81.0000 81.0000 81.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 86.0000 86.0000
86.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.0000 83.0000 83.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 81.0000 81.0000 81.0000 0.0000
0.0000 0.0000
17.0000 17.0000 17.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 87.0000
87.0000 87.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 81.0000 81.0000
81.0000 0.0000 0.0000 0.0000
17.0000 17.0000 17.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.7000 83.7000
83.7000 0.0000 0.0000 0.0000
96.0000 96.0000 96.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 94.0000 94.0000 94.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.8000 83.8000
83.8000 0.0000 0.0000 0.0000

97.0000 97.0000 97.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 95.0000 95.0000 95.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.8000 83.8000
83.8000 0.0000 0.0000 0.0000
97.0000 97.0000 97.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 95.0000 95.0000 95.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.8000 83.8000
83.8000 0.0000 0.0000 0.0000
105.0000 105.0000 105.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 95.0000 95.0000 95.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.8000 83.8000
83.8000 0.0000 0.0000 0.0000
110.0000 110.0000 110.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 88.0000
88.0000 88.0000 95.0000 95.0000 95.0000 0.0000 0.0000 0.0000 83.0000 83.0000
83.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 83.8000 83.8000
83.8000 0.0000 0.0000 0.0000

```
/*****************************************
 * tOps.animo
 * Test for operators
 * Author: Joshua Poritz
 *****************************************/

include "test.sbvh";

a = 3;
if (a == 3)
    print(1);
else
    print(0);

if (a > 4)
    print(1);
else
    print(0);

if (a < 4)
    print(1);
else
    print(0);

if (a != 3)
    print(1);
else
    print(0);

if (a <= 2)
    print(1);
else
    print(0);

if (a >= 2)
    print(1);
else
    print(0);

print(0 || 0);
print(0 || 1);
print(1 || 0);
print(1 || 1);
print(-1.25 || -1.25);

print(0 && 0);
print(1 && 0);
print(0 && 1);
print(1 && 1);
print(-1.25 && -1.25);

print(-a);
print(a + 1);
print(1 - a);
print(a * a);
print(a / a);
```

```
a += 1.5;
print(a);
a -= 1.5;
print(a);
a *= 2;
print(a);
a %= 4;
print(a);
a /= (2 / 3);
print(a);

print(!(a));
print(!(a * 0));
print(++a);
print(--a);
```

```
/************************************
 * tRotMov.animo
 * Test for rotate and move functions
 * Author: Joshua Poritz
 ************************************/

include "rotMovTest.sbvh";

// rotate and move commands with whole number start time and duration arguments
rotate(LeftKnee, 0, 4, 30, 45, 60);
rotate(LeftWrist, 0, 1, 5, 10, 10);
move(2, 3, 70, 20, 5);

// Rotation/movement with negative and fractional (x, y, z)
rotate(LeftKnee, 4, 4, -30, -45, -60);
rotate(LeftWrist, 1, 1, -5, -10.5, -10.5);
move(5, 3, -70, -20.5, -5.5);

// rotate and move commands with fractional start time and/or duration
for (i = 1 to 10)
    rotate(RightElbow, curtime(), 0.15 * i, i * 2, i * 5, i * 10);
for (i = 1 to 10)
    move(curtime(), 0.15 * i, i * 2, i * 5, i * 10);
x = ceil(curtime());
rotate(LeftShoulder, x + 0.2, 0.2, 25, 25, 25);
rotate(LeftShoulder, x + 0.4, 0.6, 30, 30, 30);

// rotates and moves overlapping some frames
rotate(RightShoulder, 15, 5, 64, 64, 64);
rotate(RightShoulder, 18, 5, 73, 73, 73);
move(15, 5, 64, 64, 64);
move(18, 5, 73, 73, 73);

// rotates and moves COMPLETELY overlapping the same frames
rotate(LeftCollar, 25, 5, 80, 80, 80);
rotate(LeftCollar, 25, 5, 90, 90, 90);
move(25, 5, 80, 80, 80);
move(25, 5, 90, 90, 90);

// rotate with invalid joint name
/*
rotate(LeftNeck, 42, 1, 60, 60, 60);
*/

// rotates and moves with zero durations
/*
rotate(Neck, 42, 0, 60, 60, 60);
move(42, 0, 60, 60, 60);
*/

// rotates and moves with negative durations and start times
/*
rotate(Neck, 42, -1, 60, 60, 60);
move(Neck, -1, 60, 60, 60);
rotate(Neck, -0.5, 1, 60, 60, 60);
move(-0.5, 1, 60, 60, 60);
*/
```

```
// rotates and moves with wrong numbers of parameters
/*
rotate();
rotate(Neck);
rotate(Neck, 4, 7);
rotate(Neck, 4, 7, 0, 0, 0, 0);
move();
move(4, 7);
move(4, 7, 0, 0, 0, 0);
*/

// rotates and moves with invalid parameters
/*
a = 3;
rotate(5, 10, 1, 0, 0, 0);
rotate(a, 10, 1, 0, 0, 0);
rotate(Neck, Neck, 1, 0, 0, 0);
rotate(Neck, 10, Neck, 0, 0, 0);
rotate(Neck, 10, 1, Neck, 0, 0);
rotate(Neck, 10, 1, 0, Neck, 0);
rotate(Neck, 10, 1, 0, 0, Neck);
move(Neck, 1, 0, 0, 0);
move(10, Neck, 0, 0, 0);
move(10, 1, Neck, 0, 0);
move(10, 1, 0, Neck, 0);
move(10, 1, 0, 0, Neck);
*/

// scheduling a rotation way ahead of time
rotate(RightCollar, 50, 10, 120, 120, 120);

// scheduling another rotation in between
rotate(RightCollar, 35, 5, 40, 40, 40);
```

```
/****************************
 * tRotMove.bvh
 ****************************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 300
Frame Time: 0.2000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.5000 -14.3750
3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.0000 -5.6000 2.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 -11.2500
6.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 2.0000 -1.7000 4.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 4.5000 -8.1250
9.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 3.0000 2.2000 6.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 6.0000 -5.0000
12.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 4.0000 6.1000 8.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 7.5000 -1.8750
15.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 5.0000 10.0000 10.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 9.0000 1.2500
18.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 3.0000 5.9000 5.9000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 10.5000 4.3750
21.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.0000 1.8000 1.8000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 7.5000
24.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -1.0000 -2.3000 -2.3000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 13.5000 10.6250
27.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -3.0000 -6.4000 -6.4000 0.0000 0.0000 0.0000 0.0000 0.0000
```

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 13.7500
30.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
4.6667 1.3333 0.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 16.5000 16.8750
33.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
9.3333 2.6667 0.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 18.0000 20.0000
36.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
14.0000 4.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 19.5000 23.1250
39.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
18.6667 5.3333 1.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 21.0000 26.2500
42.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
23.3333 6.6667 1.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 22.5000 29.3750
45.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
28.0000 8.0000 2.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 24.0000 32.5000
48.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
32.6667 9.3333 2.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 25.5000 35.6250
51.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
37.3333 10.6667 2.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 27.0000 38.7500
54.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000

42.0000 12.0000 3.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.5000 41.8750
57.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
46.6667 13.3333 3.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 30.0000 45.0000
60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
51.3333 14.6667 3.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 27.0000 40.5000
54.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
56.0000 16.0000 4.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 24.0000 36.0000
48.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
60.6667 17.3333 4.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 21.0000 31.5000
42.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
65.3333 18.6667 4.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 18.0000 27.0000
36.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
70.0000 20.0000 5.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 22.5000
30.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
60.6667 17.3000 4.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 18.0000
24.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
51.3333 14.6000 3.6000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 9.0000 13.5000
18.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
42.0000 11.9000 2.9000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 6.0000 9.0000
12.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
32.6667 9.2000 2.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.0000 4.5000
6.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
23.3333 6.5000 1.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
14.0000 3.8000 0.8000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -3.0000 -4.5000
-6.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
4.6667 1.1000 0.1000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -6.0000 -9.0000 -
12.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000
-4.6667 -1.6000 -0.6000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -9.0000 -
13.5000 -18.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-14.0000 -4.3000 -1.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -12.0000 -
18.0000 -24.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-23.3333 -7.0000 -2.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -15.0000 -
22.5000 -30.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-32.6667 -9.7000 -2.7000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -18.0000 -
27.0000 -36.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-42.0000 -12.4000 -3.4000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -21.0000 -
31.5000 -42.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000

-51.3333 -15.1000 -4.1000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -24.0000 -
36.0000 -48.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-60.6667 -17.8000 -4.8000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -27.0000 -
40.5000 -54.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 2.6667 6.6667 13.3333 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 3.5556 8.8889 17.7778 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 4.6420 11.6049 23.2099 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 5.7284 14.3210 28.6420 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 6.8148 17.0370 34.0741 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 7.2099 18.0247 36.0494 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 7.6049 19.0123 38.0247 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 8.0000 20.0000 40.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 8.5333 21.3333 42.6667 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 9.0667 22.6667 45.3333 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 9.6000 24.0000 48.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 10.1333 25.3333 50.6667 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 10.6667 26.6667 53.3333 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 11.2000 28.0000 56.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 11.7333 29.3333 58.6667 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 12.2667 30.6667 61.3333 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000

-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 12.5968 31.4921 62.9841 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 12.9270 32.3175 64.6349 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 13.2571 33.1429 66.2857 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 13.5873 33.9683 67.9365 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 13.9175 34.7937 69.5873 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 14.2646 35.6614 71.3228 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 14.6116 36.5291 73.0582 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 14.9587 37.3968 74.7937 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 15.3058 38.2646 76.5291 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 15.6529 39.1323 78.2646 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 16.0000 40.0000 80.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 16.2963 40.7407 81.4815 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 16.5926 41.4815 82.9630 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 16.8889 42.2222 84.4444 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 17.1852 42.9630 85.9259 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 17.4815 43.7037 87.4074 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 17.7778 44.4444 88.8889 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 18.0741 45.1852 90.3704 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-70.0000 -20.5000 -5.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 18.3309 45.8272 91.6543 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000

-64.6400 -17.1200 -2.7200 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
2.5600 2.5600 2.5600 18.5877 46.4691 92.9383 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-59.2800 -13.7400 0.0600 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
5.1200 5.1200 5.1200 18.8444 47.1111 94.2222 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-53.9200 -10.3600 2.8400 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
7.6800 7.6800 7.6800 19.1012 47.7531 95.5062 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
-48.5600 -6.9800 5.6200 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
10.2400 10.2400 10.2400 19.3580 48.3951 96.7901 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-43.2000 -3.6000 8.4000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
12.8000 12.8000 12.8000 19.6148 49.0370 98.0741 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-37.8400 -0.2200 11.1800 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
15.3600 15.3600 15.3600 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-32.4800 3.1600 13.9600 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
17.9200 17.9200 17.9200 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-27.1200 6.5400 16.7400 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
20.4800 20.4800 20.4800 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-21.7600 9.9200 19.5200 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
23.0400 23.0400 23.0400 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-16.4000 13.3000 22.3000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
25.6000 25.6000 25.6000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-11.0400 16.6800 25.0800 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
28.1600 28.1600 28.1600 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-5.6800 20.0600 27.8600 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
30.7200 30.7200 30.7200 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
-0.3200 23.4400 30.6400 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
33.2800 33.2800 33.2800 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
5.0400 26.8200 33.4200 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
35.8400 35.8400 35.8400 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
10.4000 30.2000 36.2000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
38.4000 38.4000 38.4000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
12.9040 31.9120 37.6720 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
39.7840 39.7840 39.7840 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
15.4080 33.6240 39.1440 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
41.1680 41.1680 41.1680 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
17.9120 35.3360 40.6160 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
42.5520 42.5520 42.5520 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
20.4160 37.0480 42.0880 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
43.9360 43.9360 43.9360 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000

22.9200 38.7600 43.5600 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
45.3200 45.3200 45.3200 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
25.4240 40.4720 45.0320 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
46.7040 46.7040 46.7040 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
27.9280 42.1840 46.5040 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
48.0880 48.0880 48.0880 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
30.4320 43.8960 47.9760 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
49.4720 49.4720 49.4720 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
32.9360 45.6080 49.4480 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
50.8560 50.8560 50.8560 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
35.4400 47.3200 50.9200 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
52.2400 52.2400 52.2400 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
37.9440 49.0320 52.3920 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
53.6240 53.6240 53.6240 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
40.4480 50.7440 53.8640 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
55.0080 55.0080 55.0080 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
42.9520 52.4560 55.3360 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
56.3920 56.3920 56.3920 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
45.4560 54.1680 56.8080 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
57.7760 57.7760 57.7760 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
47.9600 55.8800 58.2800 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
59.1600 59.1600 59.1600 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
50.4640 57.5920 59.7520 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
60.5440 60.5440 60.5440 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
52.9680 59.3040 61.2240 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
61.9280 61.9280 61.9280 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
55.4720 61.0160 62.6960 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
63.3120 63.3120 63.3120 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
57.9760 62.7280 64.1680 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
64.6960 64.6960 64.6960 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
60.4800 64.4400 65.6400 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
66.0800 66.0800 66.0800 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
62.9840 66.1520 67.1120 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
67.4640 67.4640 67.4640 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
65.4880 67.8640 68.5840 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
68.8480 68.8480 68.8480 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
67.9920 69.5760 70.0560 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
70.2320 70.2320 70.2320 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000

70.4960 71.2880 71.5280 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
71.6160 71.6160 71.6160 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
18.3309 45.8272 91.6543 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
18.5877 46.4691 92.9383 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
18.8444 47.1111 94.2222 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
19.1012 47.7531 95.5062 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
19.3580 48.3951 96.7901 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
19.6148 49.0370 98.0741 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
20.1284 50.3210 100.6420 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
20.1284 50.3210 100.6420 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
20.1284 50.3210 100.6420 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
22.9233 51.9081 100.2163 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 3.6000 3.6000 3.6000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
25.7181 53.4953 99.7906 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 7.2000 7.2000 7.2000 25.0000 25.0000
25.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000 0.0000
73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000
28.5130 55.0825 99.3649 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 10.8000 10.8000 10.8000 26.6667
26.6667 26.6667 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
31.3079 56.6696 98.9393 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 14.4000 14.4000 14.4000 28.3333
28.3333 28.3333 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
34.1027 58.2568 98.5136 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 18.0000 18.0000 18.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
36.8976 59.8440 98.0879 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 21.6000 21.6000 21.6000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
39.6924 61.4311 97.6622 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 25.2000 25.2000 25.2000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

42.4873 63.0183 97.2365 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 28.8000 28.8000 28.8000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
45.2822 64.6054 96.8109 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 32.4000 32.4000 32.4000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
48.0770 66.1926 96.3852 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 36.0000 36.0000 36.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
50.8719 67.7798 95.9595 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 39.6000 39.6000 39.6000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
53.6668 69.3669 95.5338 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 43.2000 43.2000 43.2000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
56.4616 70.9541 95.1081 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 46.8000 46.8000 46.8000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
59.2565 72.5412 94.6825 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 50.4000 50.4000 50.4000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
62.0514 74.1284 94.2568 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 54.0000 54.0000 54.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
64.8462 75.7156 93.8311 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 57.6000 57.6000 57.6000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
67.6411 77.3027 93.4054 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 61.2000 61.2000 61.2000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
70.4360 78.8899 92.9798 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 64.8000 64.8000 64.8000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
73.2308 80.4770 92.5541 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 68.4000 68.4000 68.4000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
76.0257 82.0642 92.1284 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 72.0000 72.0000 72.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
78.8205 83.6514 91.7027 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 75.6000 75.6000 75.6000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
81.6154 85.2385 91.2770 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 79.2000 79.2000 79.2000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
84.4103 86.8257 90.8514 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 82.8000 82.8000 82.8000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
87.2051 88.4128 90.4257 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 86.4000 86.4000 86.4000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 0.0000 0.0000
0.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 1.6000 1.6000
1.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 3.2000 3.2000
3.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 4.8000 4.8000
4.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 6.4000 6.4000
6.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 8.0000 8.0000
8.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 9.6000 9.6000
9.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 11.2000 11.2000
11.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 12.8000 12.8000
12.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 14.4000 14.4000
14.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 16.0000 16.0000
16.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 17.6000 17.6000
17.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 19.2000 19.2000
19.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 20.8000 20.8000
20.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 22.4000 22.4000
22.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 24.0000 24.0000
24.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 25.6000 25.6000
25.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 27.2000 27.2000
27.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 28.8000 28.8000
28.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 30.4000 30.4000
30.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 32.0000 32.0000
32.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 33.6000 33.6000
33.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 35.2000 35.2000
35.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 36.8000 36.8000
36.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 38.4000 38.4000
38.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.0000 40.0000
40.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 2.4000 2.4000
2.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 4.8000 4.8000
4.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 7.2000 7.2000
7.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 9.6000 9.6000
9.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 12.0000 12.0000
12.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 14.4000 14.4000
14.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 16.8000 16.8000
16.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 19.2000 19.2000
19.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 21.6000 21.6000
21.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 24.0000 24.0000
24.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 26.4000 26.4000
26.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 28.8000 28.8000
28.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 31.2000 31.2000
31.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 33.6000 33.6000
33.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 36.0000 36.0000
36.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 38.4000 38.4000
38.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 40.8000 40.8000
40.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 43.2000 43.2000
43.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 45.6000 45.6000
45.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 48.0000 48.0000
48.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 50.4000 50.4000
50.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 52.8000 52.8000
52.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 55.2000 55.2000
55.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 57.6000 57.6000
57.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 60.0000 60.0000
60.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 62.4000 62.4000
62.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 64.8000 64.8000
64.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 67.2000 67.2000
67.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 69.6000 69.6000
69.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 72.0000 72.0000
72.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 74.4000 74.4000
74.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 76.8000 76.8000
76.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 79.2000 79.2000
79.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 81.6000 81.6000
81.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 84.0000 84.0000
84.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 86.4000 86.4000
86.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 88.8000 88.8000
88.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 91.2000 91.2000
91.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 93.6000 93.6000
93.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 96.0000 96.0000
96.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 98.4000 98.4000
98.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 100.8000 100.8000
100.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 103.2000 103.2000
103.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000

30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 105.6000 105.6000
105.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 108.0000 108.0000
108.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 110.4000 110.4000
110.4000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 112.8000 112.8000
112.8000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 115.2000 115.2000
115.2000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 117.6000 117.6000
117.6000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
90.0000 90.0000 90.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -30.0000 -
45.0000 -60.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 90.0000 90.0000 90.0000 30.0000
30.0000 30.0000 0.0000 0.0000 0.0000 -5.0000 -10.5000 -10.5000 120.0000 120.0000
120.0000 73.0000 73.0000 73.0000 19.8716 49.6790 99.3580 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

```
/*****************************************
 * tScope.animo
 * Test for scoping
 * Author: Joshua Poritz
 *****************************************/

include "test.sbvh";

// Use of floating point declared outside a block inside the block
a = 5;
if (a == 5)
    rotate(Neck, 0, 1, a, a, a);

// Changing the floating point, then re-using it
a = 6;
rotate(Neck, 1, 1, a, a, a);

// Changing a floating point inside a block
a = 8;
if (a == 8)
{
    a = 7;
    rotate(Neck, 2, 1, a, a, a);
}
rotate(Neck, 3, 1, a, a, a);

// Using a floating point declared inside a block outside the block
/*
if (1)
{
    rotate(Neck, 4, 1, a + 1, a + 1, a + 1);
    b = 10;
}
rotate(Neck, 5, 1, b, b, b);
*/

// Use of for loop counter
for (cnt = 11 to 14)
    rotate(Neck, cnt, 1, cnt, cnt, cnt);
/* rotate(Neck, cnt + 1, 1, cnt + 1, cnt + 1, cnt + 1); */

// Demonstrate that aliases for BVH joints are global
if (1)
    n = Neck;
rotate(n, 15, 1, 15, 15, 15);

// Changing the variable type (should be disallowed)
/*
n = 16;
rotate(Neck, n, 1, n, n, n);
m = 15;
m = Neck;
*/

// More complex example testing scope pretty comprehensively
func a(x)
{
```

```
   return x;
}
/* blah = x; */  /* Trying to use variable after its scope */
func b(y)
{
    p = 23;
    move(y, 1, y, y, y);
    y = 20;
    for (i = a(6) + 10 to a(7) + 10)
    {
        j = Head;
        rotate(Neck, i, 1, i, i, i);
        if (1)
          y = 19;
      rotate(Neck, 21 + i - 16, 1, y, 21, i);
    }
    /* blah = i; */  /* Trying to use loop counter after its scope */
    rotate(j, p, 1, p, p, y);
}
b(24);
```

```
/*********************************
 * tScope.bvh
 ********************************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 25
Frame Time: 1.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 5.0000 5.0000 5.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 6.0000 6.0000 6.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 7.0000 7.0000 7.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 8.0000 8.0000 8.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 11.0000 11.0000 11.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 12.0000 12.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 13.0000 13.0000 13.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 14.0000 14.0000 14.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 15.0000 15.0000 15.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 16.0000 16.0000 16.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 17.0000 17.0000 17.0000 0.0000
0.0000 0.0000

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 17.0000 17.0000 17.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 17.0000 17.0000 17.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 17.0000 17.0000 17.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 20.0000 21.0000 16.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 20.0000 21.0000 17.0000 0.0000
0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 20.0000 21.0000 17.0000 23.0000
23.0000 20.0000
24.0000 24.0000 24.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 20.0000 21.0000 17.0000 23.0000
23.0000 20.0000

```
/*******************************************************/
/* walkfunc.animo: Sample Program */
/* Step with left leg, accompanied with left hand motion */
/* Author: Alexei Masterov */
/*******************************************************/

func left_step ( begin_frame, duration )
{
   // each step is 3 keyframes, if the user wants more -> interpolate
   step_gap =  duration / 3;

   // left hip
   rotate ( LeftHip, begin_frame, step_gap, -0.11, -10.12, 4.22 );
   rotate ( LeftHip, begin_frame + step_gap, step_gap, -1.01, -28.21, -1.56 );
   rotate ( LeftHip, begin_frame + step_gap * 2, step_gap, 4.62, -22.28, 3.55 );
   rotate ( LeftHip, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // left knee
   rotate ( LeftKnee, begin_frame, step_gap, 0, 28.79, 5.19 );
   rotate ( LeftKnee, begin_frame + step_gap, step_gap, 0, 71.64, 4.87 );
   rotate ( LeftKnee, begin_frame + step_gap * 2, step_gap, 0.01, 70.77, 5.58 );
   rotate ( LeftKnee, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // left ankle
   rotate ( LeftAnkle, begin_frame, step_gap, -0.75, -8.44, -8.67 );
   rotate ( LeftAnkle, begin_frame + step_gap, step_gap, -16.91, -17.59, -1.84
);
   rotate ( LeftAnkle, begin_frame + step_gap * 2, step_gap, -24.07, -10.11, -
7.22 );
   rotate ( LeftAnkle, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // left shoulder
   rotate ( LeftShoulder, begin_frame, step_gap, 29.83, -33.3, -46.9 );
   rotate ( LeftShoulder, begin_frame + step_gap, step_gap, 32.48, -51.08, -
62.65 );
   rotate ( LeftShoulder, begin_frame + step_gap * 2, step_gap, 28.23, -55.95, -
80.9 );
   rotate ( LeftShoulder, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // left elbow
   rotate ( LeftElbow, begin_frame, step_gap, 0, -66.72, -11.11 );
   rotate ( LeftElbow, begin_frame + step_gap, step_gap, 179.97, -89.22, 174.15
);
   rotate ( LeftElbow, begin_frame + step_gap * 2, step_gap, 0.02, -77.32, 5.77
);
   rotate ( LeftElbow, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // left wrist
   rotate ( LeftWrist, begin_frame, step_gap, -31.45, 10.67, 26.4 );
   rotate ( LeftWrist, begin_frame + step_gap, step_gap, -14.81, 9.37, 17.29 );
   rotate ( LeftWrist, begin_frame + step_gap * 2, step_gap, -10.55, -4.56, -
17.88 );
   rotate ( LeftWrist, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );
}

/*******************************************************/
/* Step with right leg, accompanied with right hand motion */
```

```
/*********************************************************/

func right_step ( begin_frame, duration )
{
   // each step is 3 keyframes, if the user wants more -> interpolate
   step_gap = duration / 3;

   // right hip
   rotate ( RightHip, begin_frame, step_gap, -5.5, -8.03, -2.54 );
   rotate ( RightHip, begin_frame + step_gap, step_gap, -2.69, -17.24, -0.31 );
   rotate ( RightHip, begin_frame + step_gap * 2, step_gap, -0.71, -28.86, -2.08
);
   rotate ( RightHip, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // right knee
   rotate ( RightKnee, begin_frame, step_gap, 0, 39.65, -5.88 );
   rotate ( RightKnee, begin_frame + step_gap, step_gap, 0, 57.12, -3.86 );
   rotate ( RightKnee, begin_frame + step_gap * 2, step_gap, 0, 71.07, -3.65 );
   rotate ( RightKnee, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // right ankle
   rotate ( RightAnkle, begin_frame, step_gap, 3.44, -18.18, 10.08 );
   rotate ( RightAnkle, begin_frame + step_gap, step_gap, 8.74, -17.5, 4.45 );
   rotate ( RightAnkle, begin_frame + step_gap * 2, step_gap, 13.63, -8.64, 2.02
);
   rotate ( RightAnkle, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // right shoulder
   rotate ( RightShoulder, begin_frame, step_gap, -36.38, -5.37, 24.88 );
   rotate ( RightShoulder, begin_frame + step_gap, step_gap, -31.19, -23.73,
47.08 );
   rotate ( RightShoulder, begin_frame + step_gap * 2, step_gap, -28.02, -40.89,
58.39 );
   rotate ( RightShoulder, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // right elbow
   rotate ( RightElbow, begin_frame, step_gap, 0, -22.49, 0.9 );
   rotate ( RightElbow, begin_frame + step_gap, step_gap, 0.01, -79.93, 5.61 );
   rotate ( RightElbow, begin_frame + step_gap * 2, step_gap, -0.03, -72.54, -
3.64 );
   rotate ( RightElbow, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );

   // right wrist
   rotate ( RightWrist, begin_frame, step_gap, 30.19, -7.37, 22.78 );
   rotate ( RightWrist, begin_frame, step_gap,23.47, 21.63, -13.6 );
   rotate ( RightWrist, begin_frame + step_gap * 2, step_gap, 16.12, 7.85, 7.15
);
   rotate ( RightWrist, begin_frame + step_gap * 3, 0.3, 0, 0, 0 );
}
```

```
/************************************************
 * wavefunc.animo: Sample program
 * Author: Alexei Masterov
 ************************************************/

func wave_right_hand (start_frame, duration)
{
      print ( duration );
      /* frame #1 */
      rotate(RightCollar, start_frame, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame, duration, -23.07, -24.76, -2.05);
      rotate(RightElbow, start_frame, duration, 0, -36.92, -1.72);
      rotate(RightWrist, start_frame, duration, 14.23, 34.21, -9.25);
      /* frame #2 */
      rotate(RightCollar, start_frame + 1, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 1, duration, -116.21, -7.74, -67.66);
      rotate(RightElbow, start_frame + 1, duration, 0, -46.8, -1.59);
      rotate(RightWrist, start_frame + 1, duration, -18.5, -32.15, 53.17);
      /* frame #3 */
      rotate(RightCollar, start_frame + 2, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 2, duration, -117.97, -3.5, -61.38);
      rotate(RightElbow, start_frame + 2, duration, 0, -38.16, -3.54);
      rotate(RightWrist, start_frame + 2, duration, -28.48, -68.93, 52.65);
      /* frame #4 */
      rotate(RightCollar, start_frame + 3, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 3, duration, -113.31, -8.07, -64.62);
      rotate(RightElbow, start_frame + 3, duration, 0, -49.99, -4.69);
      rotate(RightWrist, start_frame + 3, duration, -6.45, -69.37, 78.99);
      /* frame #5 */
      rotate(RightCollar, start_frame + 4, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 4, duration, -63.46, -8.04, -59.67);
      rotate(RightElbow, start_frame + 4, duration, 0, -24.32, -5.44);
      rotate(RightWrist, start_frame + 4, duration, -65.13, -47.65, -25.34);
      /* frame #6 */
      rotate(RightCollar, start_frame + 5, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 5, duration, -43.71, 7.44, -37.42);
      rotate(RightElbow, start_frame + 5, duration, 0, -30.8, -0.7);
      rotate(RightWrist, start_frame + 5, duration, -22.92, 37.38, 4.35);
      /* frame #7 */
      rotate(RightCollar, start_frame + 6, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 6, duration, -13.02, -16.53, 8.8);
      rotate(RightElbow, start_frame + 6, duration, 0, -39.2, 4.17);
      rotate(RightWrist, start_frame + 6, duration, -39.09, 34.51, 50.91);
      /* frame #8 */
      rotate(RightCollar, start_frame + 7, duration, 0, 0, 0);
      rotate(RightShoulder, start_frame + 7, duration, -6.64, -21.92, 21.22);
      rotate(RightElbow, start_frame + 7, duration, 0, -32.59, -4.07);
      rotate(RightWrist, start_frame + 7, duration, -7.08, -6.72, 67.88);

}
```

```
/***********************************
 * dance.animo: Sample program
 * Author: Alexei Masterov
 ***********************************/

include "dance.sbvh";

// duration of each frame in seconds
duration = 0.2;
begin_time = 0;

// put the figure in initial position
move(begin_time, 0.01, -17.129814, 39.513103, -20.553329);

/* frame #0 */
move(begin_time + 0.01, duration - 0.01, -17.129814, 39.513103, -20.553329);
rotate(Hips, begin_time, duration, -5.145142, 7.454098, -35.545376);
rotate(LeftHip, begin_time, duration, 18.684629, -12.699254, 28.043344);
rotate(LeftKnee, begin_time, duration, 5, 14.01228, 0);
rotate(LeftAnkle, begin_time, duration, -8.197182, -8.943927, 5.615734);
rotate(RightHip, begin_time, duration, 15.74406, 0.171168, 0.018856);
rotate(RightKnee, begin_time, duration, -5, 3.507612, 0);
rotate(RightAnkle, begin_time, duration, -4.224754, 1.468446, -6.9372);
rotate(Chest, begin_time, duration, 5.154825, -0.541715, 2.757841);
rotate(LeftCollar, begin_time, duration, -0.483571, 1.530707, 0.012918);
rotate(LeftShoulder, begin_time, duration, 27.021511, 11.128203, 23.415245);
rotate(LeftElbow, begin_time, duration, -4, -21.37571, 0);
rotate(LeftWrist, begin_time, duration, -22.103354, 2.79287, -33.793991);
rotate(RightCollar, begin_time, duration, -0.483571, 1.530707, 0.012918);
rotate(RightShoulder, begin_time, duration, -31.89337, 12.959333, 57.074883);
rotate(RightElbow, begin_time, duration, 4, -80.039268, 0);
rotate(RightWrist, begin_time, duration, 4.898635, -38.379738, 58.616867);
rotate(Neck, begin_time, duration, 6.651874, 18.752653, -1.721906);
rotate(Head, begin_time, duration, -31.482822, -9.030337, 35.319271);

/* frame #1 */
move(begin_time+duration, duration, -16.701021, 39.610031, -20.51726);
rotate(Hips, begin_time+duration, duration, -5.724495, 7.937147, -35.742962);
rotate(LeftHip, begin_time+duration, duration, 18.322882, -12.27552, 28.749619);
rotate(LeftKnee, begin_time+duration, duration, 5, 11.085456, 0);
rotate(LeftAnkle, begin_time+duration, duration, -5.98536, -3.260179, 5.566837);
rotate(RightHip, begin_time+duration, duration, 15.824615, 0.172553, 0.34088);
rotate(RightKnee, begin_time+duration, duration, -5, 4.094175, 0);
rotate(RightAnkle, begin_time+duration, duration, -4.022447, 0.901529, -
6.397221);
rotate(Chest, begin_time+duration, duration, 5.395926, -0.379808, 2.282838);
rotate(LeftCollar, begin_time+duration, duration, 0.644868, 1.85291, -0.020852);
rotate(LeftShoulder, begin_time+duration, duration, 14.775752, 7.906818,
14.83617);
rotate(LeftElbow, begin_time+duration, duration, -4, -22.547981, 0);
rotate(LeftWrist, begin_time+duration, duration, -29.504467, 4.250135, -
34.309799);
rotate(RightCollar, begin_time+duration, duration, 0.644868, 1.85291, -
0.020852);
rotate(RightShoulder, begin_time+duration, duration, -27.681364, 10.861183,
55.300331);
rotate(RightElbow, begin_time+duration, duration, 4, -78.663719, 0);
```

```
rotate(RightWrist, begin_time+duration, duration, 0.312259, -42.018234,
56.770298);
rotate(Neck, begin_time+duration, duration, 6.407799, 18.573019, -1.658621);
rotate(Head, begin_time+duration, duration, -31.783743, -9.183754, 34.867287);

/* frame #2 */
move(begin_time+duration*2, duration, -16.436958, 39.467709, -20.750069);
rotate(Hips, begin_time+duration*2, duration, -5.682179, 8.576076, -36.69598);
rotate(LeftHip, begin_time+duration*2, duration, 19.26314, -13.699969,
29.574968);
rotate(LeftKnee, begin_time+duration*2, duration, 5, 11.926646, 0);
rotate(LeftAnkle, begin_time+duration*2, duration, -5.939101, 0.64974,
5.577283);
rotate(RightHip, begin_time+duration*2, duration, 15.670573, -0.224643,
0.90633);
rotate(RightKnee, begin_time+duration*2, duration, -5, 3.155029, 0);
rotate(RightAnkle, begin_time+duration*2, duration, -3.106385, 1.047569, -
6.190315);
rotate(Chest, begin_time+duration*2, duration, 5.230749, 0.908727, 2.196691);
rotate(LeftCollar, begin_time+duration*2, duration, 0.322572, 2.497519, -
0.014057);
rotate(LeftShoulder, begin_time+duration*2, duration, 10.100171, 5.033052,
6.380805);
rotate(LeftElbow, begin_time+duration*2, duration, -4, -23.067116, 0);
rotate(LeftWrist, begin_time+duration*2, duration, -27.581787, 4.015338, -
39.103401);
rotate(RightCollar, begin_time+duration*2, duration, 0.322572, 2.497519, -
0.014057);
rotate(RightShoulder, begin_time+duration*2, duration, -26.958368, 9.854655,
54.975403);
rotate(RightElbow, begin_time+duration*2, duration, 4, -77.508209, 0);
rotate(RightWrist, begin_time+duration*2, duration, -2.196213, -44.679024,
56.582664);
rotate(Neck, begin_time+duration*2, duration, 6.304765, 18.206894, -1.631415);
rotate(Head, begin_time+duration*2, duration, -32.519417, -10.743377,
34.826012);
/* frame #3 */
move(begin_time+duration*3, duration, -17.44482, 39.17384, -19.847218);
rotate(Hips, begin_time+duration*3, duration, -4.301334, 10.141496, -36.106544);
rotate(LeftHip, begin_time+duration*3, duration, 21.454668, -16.970825,
31.112751);
rotate(LeftKnee, begin_time+duration*3, duration, 5, 18.638451, 0);
rotate(LeftAnkle, begin_time+duration*3, duration, -6.733837, -2.01401,
5.583443);
rotate(RightHip, begin_time+duration*3, duration, 13.825427, -0.655887,
0.967337);
rotate(RightKnee, begin_time+duration*3, duration, -5, 2.266317, 0);
rotate(RightAnkle, begin_time+duration*3, duration, -2.75584, 0.106372, -
2.390693);
rotate(Chest, begin_time+duration*3, duration, 3.055874, 2.031169, 1.959889);
rotate(LeftCollar, begin_time+duration*3, duration, 0.08067, 2.900388, -
0.004082);
rotate(LeftShoulder, begin_time+duration*3, duration, 15.606045, 2.967969,
4.837171);
rotate(LeftElbow, begin_time+duration*3, duration, -4, -27.886517, 0);
rotate(LeftWrist, begin_time+duration*3, duration, -31.164696, -0.43084, -
40.531345);
```

```
rotate(RightCollar, begin_time+duration*3, duration, 0.08067, 2.900388, -
0.004082);
rotate(RightShoulder, begin_time+duration*3, duration, -28.718487, 10.454666,
56.434525);
rotate(RightElbow, begin_time+duration*3, duration, 4, -77.251305, 0);
rotate(RightWrist, begin_time+duration*3, duration, -1.579567, -44.004166,
58.816631);
rotate(Neck, begin_time+duration*3, duration, 6.814987, 17.901215, -1.762387);
rotate(Head, begin_time+duration*3, duration, -32.1432, -13.020011, 34.309849);
/* frame #4 */
move(begin_time+duration*4, duration, -13.804417, 39.363987, -16.844353);
rotate(Hips, begin_time+duration*4, duration, -3.278627, 6.159461, -20.895088);
rotate(LeftHip, begin_time+duration*4, duration, 8.00195, -19.956074,
21.958853);
rotate(LeftKnee, begin_time+duration*4, duration, 5, 19.896765, 0);
rotate(LeftAnkle, begin_time+duration*4, duration, -1.895744, 3.342982, -
4.131546);
rotate(RightHip, begin_time+duration*4, duration, 4.755366, 1.913827, -
1.040583);
rotate(RightKnee, begin_time+duration*4, duration, -5, 11.558072, 0);
rotate(RightAnkle, begin_time+duration*4, duration, 6.70347, -9.655918, -
3.271017);
rotate(Chest, begin_time+duration*4, duration, 7.79899, 2.36912, 2.125154);
rotate(LeftCollar, begin_time+duration*4, duration, -2.497621, 0.402449,
0.017554);
rotate(LeftShoulder, begin_time+duration*4, duration, 23.418095, -3.267248,
32.554077);
rotate(LeftElbow, begin_time+duration*4, duration, -4, -33.269577, 0);
rotate(LeftWrist, begin_time+duration*4, duration, -0.384012, 8.584818,
6.856423);
rotate(RightCollar, begin_time+duration*4, duration, -2.497621, 0.402449,
0.017554);
rotate(RightShoulder, begin_time+duration*4, duration, -29.005827, 17.24505,
34.535248);
rotate(RightElbow, begin_time+duration*4, duration, 4, -70.186607, 0);
rotate(RightWrist, begin_time+duration*4, duration, -10.889338, -40.83585,
50.585796);
rotate(Neck, begin_time+duration*4, duration, 2.191028, 17.524885, -0.567506);
rotate(Head, begin_time+duration*4, duration, -21.379515, -4.004979, 20.406166);
/* frame #5 */
move(begin_time+duration*5, duration, -12.913258, 36.754375, -15.254353);
rotate(Hips, begin_time+duration*5, duration, 6.202761, -10.001901, 0.365909);
rotate(LeftHip, begin_time+duration*5, duration, -9.308453, -14.229591,
6.657677);
rotate(LeftKnee, begin_time+duration*5, duration, 5, 37.161026, 0);
rotate(LeftAnkle, begin_time+duration*5, duration, -2.449677, -9.497665, -
3.00928);
rotate(RightHip, begin_time+duration*5, duration, -13.667568, -14.627858, -
9.706951);
rotate(RightKnee, begin_time+duration*5, duration, -5, 75.42865, 0);
rotate(RightAnkle, begin_time+duration*5, duration, 8.84893, 1.987457,
0.395745);
rotate(Chest, begin_time+duration*5, duration, 3.273613, 13.472042, 9.141914);
rotate(LeftCollar, begin_time+duration*5, duration, -5.735439, 4.168554,
0.418305);
rotate(LeftShoulder, begin_time+duration*5, duration, 6.883256, -4.37008, -
2.815387);
```

```
rotate(LeftElbow, begin_time+duration*5, duration, -4, -72.103004, 0);
rotate(LeftWrist, begin_time+duration*5, duration, -5.323997, 8.678768, -
2.287296);
rotate(RightCollar, begin_time+duration*5, duration, -5.735439, 4.168554,
0.418305);
rotate(RightShoulder, begin_time+duration*5, duration, -9.934323, -10.115631,
0.27028);
rotate(RightElbow, begin_time+duration*5, duration, 4, -40.831001, 0);
rotate(RightWrist, begin_time+duration*5, duration, -30.275377, -3.769179,
8.136178);
rotate(Neck, begin_time+duration*5, duration, -3.990515, 16.393581, 1.032363);
rotate(Head, begin_time+duration*5, duration, 2.644384, -4.640108, -6.501391);
/* frame #6 */
move(begin_time+duration*6, duration, -13.729616, 38.91571, -14.079136);
rotate(Hips, begin_time+duration*6, duration, -4.687915, -2.721347, 3.006787);
rotate(LeftHip, begin_time+duration*6, duration, 1.750564, -14.481229,
4.332572);
rotate(LeftKnee, begin_time+duration*6, duration, 5, 25.709202, 0);
rotate(LeftAnkle, begin_time+duration*6, duration, -1.944163, -3.521772, -
9.299974);
rotate(RightHip, begin_time+duration*6, duration, -8.281845, -22.86516, -
10.094368);
rotate(RightKnee, begin_time+duration*6, duration, -5, 50.943615, 0);
rotate(RightAnkle, begin_time+duration*6, duration, 11.05654, -1.060217,
4.828407);
rotate(Chest, begin_time+duration*6, duration, 2.310683, 16.612484, 4.596787);
rotate(LeftCollar, begin_time+duration*6, duration, -2.424496, 4.507697,
0.190662);
rotate(LeftShoulder, begin_time+duration*6, duration, 6.613963, -9.679655, -
16.146868);
rotate(LeftElbow, begin_time+duration*6, duration, -4, -74.0467, 0);
rotate(LeftWrist, begin_time+duration*6, duration, 2.458456, 0.441579, -
12.047024);
rotate(RightCollar, begin_time+duration*6, duration, -2.424496, 4.507697,
0.190662);
rotate(RightShoulder, begin_time+duration*6, duration, -4.260518, -17.980421,
29.872219);
rotate(RightElbow, begin_time+duration*6, duration, 4, -72.128159, 0);
rotate(RightWrist, begin_time+duration*6, duration, -8.674229, -16.570129,
45.654068);
rotate(Neck, begin_time+duration*6, duration, -0.966407, 14.966805, 0.250113);
rotate(Head, begin_time+duration*6, duration, 4.39418, -12.313986, -1.102398);
/* frame #7 */
move(begin_time+duration*7, duration, -17.605885, 38.09581, -11.617993);
rotate(Hips, begin_time+duration*7, duration, -5.271516, -1.365486, -3.223443);
rotate(LeftHip, begin_time+duration*7, duration, 10.560674, -15.811641,
6.759228);
rotate(LeftKnee, begin_time+duration*7, duration, 5, 37.040127, 0);
rotate(LeftAnkle, begin_time+duration*7, duration, -9.443357, -10.716975, -
10.1193);
rotate(RightHip, begin_time+duration*7, duration, -14.779498, -20.62763, -
12.051445);
rotate(RightKnee, begin_time+duration*7, duration, -5, 13.870152, 0);
rotate(RightAnkle, begin_time+duration*7, duration, 5.313439, 18.43741, -
4.815678);
rotate(Chest, begin_time+duration*7, duration, 2.705829, 18.385557, 6.465448);
```

```
rotate(LeftCollar, begin_time+duration*7, duration, -3.969739, 6.02809,
0.417547);
rotate(LeftShoulder, begin_time+duration*7, duration, 11.212226, -3.13238,
0.631595);
rotate(LeftElbow, begin_time+duration*7, duration, -4, -60.894497, 0);
rotate(LeftWrist, begin_time+duration*7, duration, 3.739194, 9.478383, -
3.233555);
rotate(RightCollar, begin_time+duration*7, duration, -3.969739, 6.02809,
0.417547);
rotate(RightShoulder, begin_time+duration*7, duration, -7.50212, -18.328396,
28.859022);
rotate(RightElbow, begin_time+duration*7, duration, 4, -59.123852, 0);
rotate(RightWrist, begin_time+duration*7, duration, -7.490026, -14.430984,
47.442421);
rotate(Neck, begin_time+duration*7, duration, -1.079134, 13.942091, 0.279332);
rotate(Head, begin_time+duration*7, duration, 3.657167, -16.902508, 0.938171);
/* frame #8 */
move(begin_time+duration*8, duration, -23.600647, 37.698734, -10.166685);
rotate(Hips, begin_time+duration*8, duration, -2.185839, 0.385279, 1.215283);
rotate(LeftHip, begin_time+duration*8, duration, 17.258251, -12.420403,
7.362471);
rotate(LeftKnee, begin_time+duration*8, duration, 5, 26.835659, 0);
rotate(LeftAnkle, begin_time+duration*8, duration, -17.740488, 0.045887, -
15.781089);
rotate(RightHip, begin_time+duration*8, duration, -16.552633, -25.895638, -
13.750904);
rotate(RightKnee, begin_time+duration*8, duration, -5, 41.909149, 0);
rotate(RightAnkle, begin_time+duration*8, duration, 16.062405, 4.09843,
7.506119);
rotate(Chest, begin_time+duration*8, duration, 1.593486, 16.692188, 4.087021);
rotate(LeftCollar, begin_time+duration*8, duration, -3.068497, 3.861659,
0.206853);
rotate(LeftShoulder, begin_time+duration*8, duration, 13.304035, 3.396105,
4.504075);
rotate(LeftElbow, begin_time+duration*8, duration, -4, -61.148354, 0);
rotate(LeftWrist, begin_time+duration*8, duration, -0.880527, 10.686496, -
5.476295);
rotate(RightCollar, begin_time+duration*8, duration, -3.068497, 3.861659,
0.206853);
rotate(RightShoulder, begin_time+duration*8, duration, -8.718978, -8.713498,
23.496515);
rotate(RightElbow, begin_time+duration*8, duration, 4, -63.392948, 0);
rotate(RightWrist, begin_time+duration*8, duration, -13.404161, -5.152517,
30.957378);
rotate(Neck, begin_time+duration*8, duration, -0.359833, 13.944358, 0.093147);
rotate(Head, begin_time+duration*8, duration, -2.638163, -15.480479, 5.779716);
/* frame #9 */
move(begin_time+duration*9, duration, -31.304668, 37.535427, -10.720301);
rotate(Hips, begin_time+duration*9, duration, 6.427833, -4.060543, 0.601263);
rotate(LeftHip, begin_time+duration*9, duration, 19.330179, -28.018993,
10.517758);
rotate(LeftKnee, begin_time+duration*9, duration, 5, 60.249142, 0);
rotate(LeftAnkle, begin_time+duration*9, duration, -18.865295, 8.206586, -
23.602928);
rotate(RightHip, begin_time+duration*9, duration, -12.014886, -18.610247, -
9.646481);
rotate(RightKnee, begin_time+duration*9, duration, -5.000001, 42.457005, 0);
```

```
rotate(RightAnkle, begin_time+duration*9, duration, 15.28175, -11.541622,
14.299189);
rotate(Chest, begin_time+duration*9, duration, -1.905475, 21.768213, 3.794538);
rotate(LeftCollar, begin_time+duration*9, duration, -5.253003, 4.492848,
0.412642);
rotate(LeftShoulder, begin_time+duration*9, duration, 8.180956, 1.949255,
13.26299);
rotate(LeftElbow, begin_time+duration*9, duration, -4, -68.251106, 0);
rotate(LeftWrist, begin_time+duration*9, duration, 4.578137, 5.445683, -
0.036052);
rotate(RightCollar, begin_time+duration*9, duration, -5.253003, 4.492848,
0.412642);
rotate(RightShoulder, begin_time+duration*9, duration, -11.833835, -2.29537,
21.839031);
rotate(RightElbow, begin_time+duration*9, duration, 4, -62.177822, 0);
rotate(RightWrist, begin_time+duration*9, duration, -13.533296, -3.750202,
33.125626);
rotate(Neck, begin_time+duration*9, duration, -0.676574, 14.967844, 0.175106);
rotate(Head, begin_time+duration*9, duration, -8.830094, -16.107822, 4.833903);
/* frame #10 */
move(begin_time+duration*10, duration, -32.732391, 38.542297, -11.074305);
rotate(Hips, begin_time+duration*10, duration, 6.351815, -7.001144, -6.431266);
rotate(LeftHip, begin_time+duration*10, duration, 6.705881, -52.716469,
14.462562);
rotate(LeftKnee, begin_time+duration*10, duration, 5, 86.088631, 0);
rotate(LeftAnkle, begin_time+duration*10, duration, -25.022865, 0.534009, -
27.579309);
rotate(RightHip, begin_time+duration*10, duration, -7.90794, -14.431717, -
7.414193);
rotate(RightKnee, begin_time+duration*10, duration, -5, 33.901268, 0);
rotate(RightAnkle, begin_time+duration*10, duration, 8.193705, -5.243793,
16.185028);
rotate(Chest, begin_time+duration*10, duration, 2.369667, 28.536413, 6.702434);
rotate(LeftCollar, begin_time+duration*10, duration, -5.131217, 8.42606,
0.753865);
rotate(LeftShoulder, begin_time+duration*10, duration, 7.422952, 0.441218,
13.327086);
rotate(LeftElbow, begin_time+duration*10, duration, -4, -75.195961, 0);
rotate(LeftWrist, begin_time+duration*10, duration, 12.174783, -5.499401, -
18.963175);
rotate(RightCollar, begin_time+duration*10, duration, -5.131217, 8.42606,
0.753865);
rotate(RightShoulder, begin_time+duration*10, duration, -15.204421, -5.032105,
2.209987);
rotate(RightElbow, begin_time+duration*10, duration, 4, -58.643513, 0);
rotate(RightWrist, begin_time+duration*10, duration, -16.825235, -2.586789,
27.067694);
rotate(Neck, begin_time+duration*10, duration, -0.363799, 14.090647, 0.094169);
rotate(Head, begin_time+duration*10, duration, -20.503429, -20.385145,
8.081726);
/* frame #11 */
move(begin_time+duration*11, duration, -32.27866, 39.272934, -11.011002);
rotate(Hips, begin_time+duration*11, duration, 6.154453, -15.45161, -20.755604);
rotate(LeftHip, begin_time+duration*11, duration, 0.893926, -59.990433,
10.573915);
rotate(LeftKnee, begin_time+duration*11, duration, 5, 85.813202, 0);
```

```
rotate(LeftAnkle, begin_time+duration*11, duration, -25.48074, -2.385731, -
23.452936);
rotate(RightHip, begin_time+duration*11, duration, -8.574259, -8.349257, -
5.190028);
rotate(RightKnee, begin_time+duration*11, duration, -5, 29.429222, 0);
rotate(RightAnkle, begin_time+duration*11, duration, -0.697423, -1.757561,
9.854114);
rotate(Chest, begin_time+duration*11, duration, -0.632914, 26.520632, 9.159431);
rotate(LeftCollar, begin_time+duration*11, duration, -9.220221, 9.067887,
1.465505);
rotate(LeftShoulder, begin_time+duration*11, duration, 10.180879, 6.381096,
16.545647);
rotate(LeftElbow, begin_time+duration*11, duration, -4, -59.051235, 0);
rotate(LeftWrist, begin_time+duration*11, duration, 12.747101, 9.152156, -
5.138736);
rotate(RightCollar, begin_time+duration*11, duration, -9.220221, 9.067887,
1.465505);
rotate(RightShoulder, begin_time+duration*11, duration, -21.508057, -16.779799,
-6.544375);
rotate(RightElbow, begin_time+duration*11, duration, 4, -92.664253, 0);
rotate(RightWrist, begin_time+duration*11, duration, -21.055349, -12.317038,
26.088436);
rotate(Neck, begin_time+duration*11, duration, 3.48938, 15.035939, -0.902598);
rotate(Head, begin_time+duration*11, duration, -29.706583, -24.043907,
22.460142);
/* frame #12 */
move(begin_time+duration*12, duration, -33.316418, 36.989594, -6.949919);
rotate(Hips, begin_time+duration*12, duration, -3.084077, -23.768581, -
27.830767);
rotate(LeftHip, begin_time+duration*12, duration, 5.669065, -32.096844,
8.338368);
rotate(LeftKnee, begin_time+duration*12, duration, 5, 89.195702, 0);
rotate(LeftAnkle, begin_time+duration*12, duration, -21.951626, -4.681513, -
22.342213);
rotate(RightHip, begin_time+duration*12, duration, -7.449198, -0.625773, -
3.457067);
rotate(RightKnee, begin_time+duration*12, duration, -5, 49.150726, 0);
rotate(RightAnkle, begin_time+duration*12, duration, 4.168019, -22.077457,
13.293082);
rotate(Chest, begin_time+duration*12, duration, 5.330661, 18.794889, 10.027803);
rotate(LeftCollar, begin_time+duration*12, duration, -8.074035, 3.749193,
0.531461);
rotate(LeftShoulder, begin_time+duration*12, duration, 7.730994, 6.135199,
20.182251);
rotate(LeftElbow, begin_time+duration*12, duration, -4, -61.81641, 0);
rotate(LeftWrist, begin_time+duration*12, duration, 9.851315, 4.738596, -
7.813559);
rotate(RightCollar, begin_time+duration*12, duration, -8.074035, 3.749193,
0.531461);
rotate(RightShoulder, begin_time+duration*12, duration, -22.609959, -31.443661,
-3.00033);
rotate(RightElbow, begin_time+duration*12, duration, 4, -76.438675, 0);
rotate(RightWrist, begin_time+duration*12, duration, -19.632473, -6.424749,
16.513079);
rotate(Neck, begin_time+duration*12, duration, 3.84443, 18.51812, -0.996227);
rotate(Head, begin_time+duration*12, duration, -29.897303, -4.928665,
30.755003);
```

```
/* frame #13 */
move(begin_time+duration*13, duration, -35.012768, 38.488434, -10.228033);
rotate(Hips, begin_time+duration*13, duration, 5.212686, -9.26523, -10.699771);
rotate(LeftHip, begin_time+duration*13, duration, 2.858577, -42.437805,
7.272922);
rotate(LeftKnee, begin_time+duration*13, duration, 5, 66.438011, 0);
rotate(LeftAnkle, begin_time+duration*13, duration, -19.621138, -5.615705, -
19.89646);
rotate(RightHip, begin_time+duration*13, duration, -0.838473, -8.305875, -
3.088899);
rotate(RightKnee, begin_time+duration*13, duration, -5, 30.726648, 0);
rotate(RightAnkle, begin_time+duration*13, duration, 4.394121, -14.292957, -
2.139857);
rotate(Chest, begin_time+duration*13, duration, -1.352536, 18.786697, 5.105765);
rotate(LeftCollar, begin_time+duration*13, duration, -3.709799, 2.57273,
0.166756);
rotate(LeftShoulder, begin_time+duration*13, duration, 6.869457, -0.884872,
19.218727);
rotate(LeftElbow, begin_time+duration*13, duration, -4, -65.588669, 0);
rotate(LeftWrist, begin_time+duration*13, duration, 9.487166, 3.076603, -
12.155961);
rotate(RightCollar, begin_time+duration*13, duration, -3.709799, 2.57273,
0.166756);
rotate(RightShoulder, begin_time+duration*13, duration, -12.52783, 4.982889,
8.816996);
rotate(RightElbow, begin_time+duration*13, duration, 4, -69.73143, 0);
rotate(RightWrist, begin_time+duration*13, duration, -17.809412, -11.324987,
15.585416);
rotate(Neck, begin_time+duration*13, duration, 2.449895, 16.11396, -0.634022);
rotate(Head, begin_time+duration*13, duration, -23.467354, -9.043563, 23.81576);
/* frame #14 */
move(begin_time+duration*14, duration, -33.663204, 36.736336, -7.270027);
rotate(Hips, begin_time+duration*14, duration, -1.695279, -9.798649, -9.286841);
rotate(LeftHip, begin_time+duration*14, duration, 11.939426, -31.224306,
10.923769);
rotate(LeftKnee, begin_time+duration*14, duration, 5, 48.903454, 0);
rotate(LeftAnkle, begin_time+duration*14, duration, -15.902702, -6.770611, -
9.040105);
rotate(RightHip, begin_time+duration*14, duration, 1.939302, -9.087251, -
2.718635);
rotate(RightKnee, begin_time+duration*14, duration, -5, 44.696926, 0);
rotate(RightAnkle, begin_time+duration*14, duration, 6.471078, -20.643593,
7.620894);
rotate(Chest, begin_time+duration*14, duration, -1.26928, 22.895517, 2.448728);
rotate(LeftCollar, begin_time+duration*14, duration, -1.46241, 7.409721,
0.188638);
rotate(LeftShoulder, begin_time+duration*14, duration, 8.210942, 6.011745,
24.847458);
rotate(LeftElbow, begin_time+duration*14, duration, -4, -58.536957, 0);
rotate(LeftWrist, begin_time+duration*14, duration, 14.590768, 6.098452, -
1.076313);
rotate(RightCollar, begin_time+duration*14, duration, -1.46241, 7.409721,
0.188638);
rotate(RightShoulder, begin_time+duration*14, duration, -11.519151, 2.730728, -
5.626769);
rotate(RightElbow, begin_time+duration*14, duration, 4, -75.558502, 0);
```

```
rotate(RightWrist, begin_time+duration*14, duration, -19.310085, -18.617662,
18.274012);
rotate(Neck, begin_time+duration*14, duration, 4.756541, 15.521576, -1.229825);
rotate(Head, begin_time+duration*14, duration, -27.818129, -23.093639,
26.447117);
/* frame #15 */
move(begin_time+duration*15, duration, -31.593407, 38.429569, -0.107606);
rotate(Hips, begin_time+duration*15, duration, -7.236069, 2.736379, -4.11649);
rotate(LeftHip, begin_time+duration*15, duration, 18.453445, -24.334566,
21.183193);
rotate(LeftKnee, begin_time+duration*15, duration, 5, 29.462717, 0);
rotate(LeftAnkle, begin_time+duration*15, duration, -7.232381, 1.51006, -
8.790439);
rotate(RightHip, begin_time+duration*15, duration, 3.067137, 1.642617, -
2.35799);
rotate(RightKnee, begin_time+duration*15, duration, -5, 24.662708, 0);
rotate(RightAnkle, begin_time+duration*15, duration, 13.708294, -19.177113,
5.760456);
rotate(Chest, begin_time+duration*15, duration, 2.979552, -0.224835, -0.701676);
rotate(LeftCollar, begin_time+duration*15, duration, 2.258437, -2.737133,
0.107905);
rotate(LeftShoulder, begin_time+duration*15, duration, 8.297648, 9.200754,
24.730993);
rotate(LeftElbow, begin_time+duration*15, duration, -4, -49.402401, 0);
rotate(LeftWrist, begin_time+duration*15, duration, 11.464435, 3.255547, -
5.32217);
rotate(RightCollar, begin_time+duration*15, duration, 2.258437, -2.737133,
0.107905);
rotate(RightShoulder, begin_time+duration*15, duration, -17.064674, -16.852499,
-37.412354);
rotate(RightElbow, begin_time+duration*15, duration, 4, -103.86235, 0);
rotate(RightWrist, begin_time+duration*15, duration, -47.784214, -1.877796,
4.78383);
rotate(Neck, begin_time+duration*15, duration, 2.700413, 18.682549, -0.700135);
rotate(Head, begin_time+duration*15, duration, -20.926788, 0.008929, 26.309801);
/* frame #16 */
move(begin_time+duration*16, duration, -25.697178, 38.03899, 3.508533);
rotate(Hips, begin_time+duration*16, duration, -8.597879, -9.990374, 10.366679);
rotate(LeftHip, begin_time+duration*16, duration, 7.56803, -16.418871,
8.130315);
rotate(LeftKnee, begin_time+duration*16, duration, 5, 41.720665, 0);
rotate(LeftAnkle, begin_time+duration*16, duration, -4.262243, -10.700898, -
5.03381);
rotate(RightHip, begin_time+duration*16, duration, -2.819038, -24.394457, -
7.705383);
rotate(RightKnee, begin_time+duration*16, duration, -5, 91.21402, 0);
rotate(RightAnkle, begin_time+duration*16, duration, -1.649781, 11.472948,
19.753998);
rotate(Chest, begin_time+duration*16, duration, 0.560178, 9.923924, -4.169472);
rotate(LeftCollar, begin_time+duration*16, duration, 3.224058, -1.689218,
0.09514);
rotate(LeftShoulder, begin_time+duration*16, duration, 13.134015, 8.216235,
24.048523);
rotate(LeftElbow, begin_time+duration*16, duration, -4, -58.366673, 0);
rotate(LeftWrist, begin_time+duration*16, duration, 5.170369, 0.339265, -
6.074597);
```

```
rotate(RightCollar, begin_time+duration*16, duration, 3.224058, -1.689218,
0.09514);
rotate(RightShoulder, begin_time+duration*16, duration, -35.29644, 9.963168, -
28.3918);
rotate(RightElbow, begin_time+duration*16, duration, 4, -84.82312, 0);
rotate(RightWrist, begin_time+duration*16, duration, -5.706198, -10.867316,
3.552155);
rotate(Neck, begin_time+duration*16, duration, 2.239107, 16.654568, -0.579631);
rotate(Head, begin_time+duration*16, duration, -9.830771, -2.520955, 19.868963);
/* frame #17 */
move(begin_time+duration*17, duration, -22.647726, 39.401943, 6.762849);
rotate(Hips, begin_time+duration*17, duration, -9.14197, -21.02965, 29.310394);
rotate(LeftHip, begin_time+duration*17, duration, 8.839565, 0.531282, 0.545652);
rotate(LeftKnee, begin_time+duration*17, duration, 5, 32.148582, 0);
rotate(LeftAnkle, begin_time+duration*17, duration, 1.425214, -14.855791, -
5.115057);
rotate(RightHip, begin_time+duration*17, duration, 5.361639, -51.340919, -
5.630099);
rotate(RightKnee, begin_time+duration*17, duration, -5, 108.192879, 0);
rotate(RightAnkle, begin_time+duration*17, duration, 6.257185, -6.955666,
18.854788);
rotate(Chest, begin_time+duration*17, duration, 4.315859, 11.943705, -9.674056);
rotate(LeftCollar, begin_time+duration*17, duration, 8.218926, -0.956869,
0.138203);
rotate(LeftShoulder, begin_time+duration*17, duration, 11.437817, 9.199423,
24.217394);
rotate(LeftElbow, begin_time+duration*17, duration, -4, -55.829403, 0);
rotate(LeftWrist, begin_time+duration*17, duration, 7.666867, 4.597013, -
5.731146);
rotate(RightCollar, begin_time+duration*17, duration, 8.218926, -0.956869,
0.138203);
rotate(RightShoulder, begin_time+duration*17, duration, -23.990576, -11.461184,
10.906994);
rotate(RightElbow, begin_time+duration*17, duration, 4, -98.185417, 0);
rotate(RightWrist, begin_time+duration*17, duration, 22.987438, 1.622884,
18.847523);
rotate(Neck, begin_time+duration*17, duration, 0.71206, 18.921223, -0.184723);
rotate(Head, begin_time+duration*17, duration, 5.011434, 5.27106, -0.919087);
/* frame #18 */
move(begin_time+duration*18, duration, -19.576277, 38.724869, 9.098734);
rotate(Hips, begin_time+duration*18, duration, -4.021911, -13.947841,
42.548473);
rotate(LeftHip, begin_time+duration*18, duration, -0.105155, 4.482029, -
5.653007);
rotate(LeftKnee, begin_time+duration*18, duration, 5, 25.560541, 0);
rotate(LeftAnkle, begin_time+duration*18, duration, 1.923022, -19.407925, -
1.312229);
rotate(RightHip, begin_time+duration*18, duration, 11.252844, -44.133507, -
1.537978);
rotate(RightKnee, begin_time+duration*18, duration, -5, 49.356453, 0);
rotate(RightAnkle, begin_time+duration*18, duration, 17.826344, -6.808752,
8.581303);
rotate(Chest, begin_time+duration*18, duration, 2.482679, 8.717219, -8.613018);
rotate(LeftCollar, begin_time+duration*18, duration, 6.123101, -0.24032,
0.025781);
rotate(LeftShoulder, begin_time+duration*18, duration, 9.79945, 21.014687,
27.550228);
```

```
rotate(LeftElbow, begin_time+duration*18, duration, -4, -52.686272, 0);
rotate(LeftWrist, begin_time+duration*18, duration, 3.784122, 4.01098, -
6.963547);
rotate(RightCollar, begin_time+duration*18, duration, 6.123101, -0.24032,
0.025781);
rotate(RightShoulder, begin_time+duration*18, duration, -17.884668, 18.353355, -
60.230564);
rotate(RightElbow, begin_time+duration*18, duration, 4.000001, -91.421753, 0);
rotate(RightWrist, begin_time+duration*18, duration, -34.974564, 7.6621,
0.714494);
rotate(Neck, begin_time+duration*18, duration, -0.998303, 17.426891, 0.2586);
rotate(Head, begin_time+duration*18, duration, 15.330964, 1.089054, -8.786468);
/* frame #19 */
move(begin_time+duration*19, duration, -12.350068, 36.049206, 18.580812);
rotate(Hips, begin_time+duration*19, duration, -3.908316, -5.693626, 50.656548);
rotate(LeftHip, begin_time+duration*19, duration, -0.756879, 7.715332, -
7.54873);
rotate(LeftKnee, begin_time+duration*19, duration, 5, 41.09774, 0);
rotate(LeftAnkle, begin_time+duration*19, duration, -9.326496, -26.550417, -
12.502378);
rotate(RightHip, begin_time+duration*19, duration, 14.993808, -31.326401, -
0.211357);
rotate(RightKnee, begin_time+duration*19, duration, -5, 35.164063, 0);
rotate(RightAnkle, begin_time+duration*19, duration, 2.206248, 6.756851,
9.054711);
rotate(Chest, begin_time+duration*19, duration, -0.963585, 4.445596, -
15.663724);
rotate(LeftCollar, begin_time+duration*19, duration, 6.848312, -0.399958,
0.048034);
rotate(LeftShoulder, begin_time+duration*19, duration, 14.801471, 14.147343,
30.496218);
rotate(LeftElbow, begin_time+duration*19, duration, -4, -56.911797, 0);
rotate(LeftWrist, begin_time+duration*19, duration, 4.816602, 1.049923, -
0.127078);
rotate(RightCollar, begin_time+duration*19, duration, 6.848312, -0.399958,
0.048034);
rotate(RightShoulder, begin_time+duration*19, duration, -22.998663, 12.359883,
0.09386);
rotate(RightElbow, begin_time+duration*19, duration, 4, -93.555344, 0);
rotate(RightWrist, begin_time+duration*19, duration, 10.58118, -5.635666,
24.96076);
rotate(Neck, begin_time+duration*19, duration, -2.953969, 16.921141, 0.764666);
rotate(Head, begin_time+duration*19, duration, 13.503547, -5.643531, -
14.019279);
/* frame #20 */
move(begin_time+duration*20, duration, -7.114614, 38.435638, 19.081347);
rotate(Hips, begin_time+duration*20, duration, 1.343216, -1.574224, 56.73048);
rotate(LeftHip, begin_time+duration*20, duration, 16.064003, -29.910963,
19.32649);
rotate(LeftKnee, begin_time+duration*20, duration, 5, 91.905159, 0);
rotate(LeftAnkle, begin_time+duration*20, duration, -13.328533, -13.095946, -
21.585583);
rotate(RightHip, begin_time+duration*20, duration, 1.675439, -16.817625, -
6.558056);
rotate(RightKnee, begin_time+duration*20, duration, -5, 26.209188, 0);
rotate(RightAnkle, begin_time+duration*20, duration, 5.651016, 2.30672,
10.968217);
```

```
rotate(Chest, begin_time+duration*20, duration, 1.603339, 7.10479, -5.341287);
rotate(LeftCollar, begin_time+duration*20, duration, -0.725511, -1.933439, -
0.024479);
rotate(LeftShoulder, begin_time+duration*20, duration, 12.87215, 17.298964,
30.875441);
rotate(LeftElbow, begin_time+duration*20, duration, -4, -54.136639, 0);
rotate(LeftWrist, begin_time+duration*20, duration, 5.947587, 4.429058, -
2.786671);
rotate(RightCollar, begin_time+duration*20, duration, -0.725511, -1.933439, -
0.024479);
rotate(RightShoulder, begin_time+duration*20, duration, -17.213139, 19.578344, -
57.210838);
rotate(RightElbow, begin_time+duration*20, duration, 4, -105.109604, 0);
rotate(RightWrist, begin_time+duration*20, duration, -38.383766, -3.984361,
4.766713);
rotate(Neck, begin_time+duration*20, duration, -6.39965, 17.05987, 1.654302);
rotate(Head, begin_time+duration*20, duration, 16.146664, -6.942715, -
19.591543);
/* frame #21 */
move(begin_time+duration*21, duration, -3.568674, 36.438126, 19.670063);
rotate(Hips, begin_time+duration*21, duration, 7.446728, -2.37756, 61.941776);
rotate(LeftHip, begin_time+duration*21, duration, 22.193066, -36.092335,
34.8116);
rotate(LeftKnee, begin_time+duration*21, duration, 5, 81.862816, 0);
rotate(LeftAnkle, begin_time+duration*21, duration, -18.798332, -13.096813, -
22.381237);
rotate(RightHip, begin_time+duration*21, duration, -6.312239, -13.925522, -
8.365907);
rotate(RightKnee, begin_time+duration*21, duration, -5, 47.359772, 0);
rotate(RightAnkle, begin_time+duration*21, duration, 15.8466, -18.616632,
20.380396);
rotate(Chest, begin_time+duration*21, duration, 4.708108, 14.361793, 2.104641);
rotate(LeftCollar, begin_time+duration*21, duration, -1.210133, 2.980293,
0.062927);
rotate(LeftShoulder, begin_time+duration*21, duration, 11.786359, 17.675713,
36.412388);
rotate(LeftElbow, begin_time+duration*21, duration, -4, -59.46727, 0);
rotate(LeftWrist, begin_time+duration*21, duration, 3.032277, -0.846374, -
3.904777);
rotate(RightCollar, begin_time+duration*21, duration, -1.210133, 2.980293,
0.062927);
rotate(RightShoulder, begin_time+duration*21, duration, -13.366383, 19.284698, -
18.468691);
rotate(RightElbow, begin_time+duration*21, duration, 4, -76.635895, 0);
rotate(RightWrist, begin_time+duration*21, duration, -8.960938, 10.94156, -
1.469352);
rotate(Neck, begin_time+duration*21, duration, -8.036469, 14.953121, 2.073633);
rotate(Head, begin_time+duration*21, duration, -0.888819, -8.378451, -
10.934739);
/* frame #22 */
move(begin_time+duration*22, duration, -0.406935, 38.298141, 21.386162);
rotate(Hips, begin_time+duration*22, duration, -3.718359, -1.211248, 55.594067);
rotate(LeftHip, begin_time+duration*22, duration, 9.228383, -22.645313,
24.767464);
rotate(LeftKnee, begin_time+duration*22, duration, 5, 25.112, 0);
rotate(LeftAnkle, begin_time+duration*22, duration, -5.572019, -4.743712, -
9.686931);
```

```
rotate(RightHip, begin_time+duration*22, duration, 0.789116, -8.213847, -
6.313642);
rotate(RightKnee, begin_time+duration*22, duration, -5, 24.619989, 0);
rotate(RightAnkle, begin_time+duration*22, duration, 12.725786, -7.95137,
20.842644);
rotate(Chest, begin_time+duration*22, duration, 6.339537, 5.504952, 4.170451);
rotate(LeftCollar, begin_time+duration*22, duration, -6.44554, -0.480343, -
0.054265);
rotate(LeftShoulder, begin_time+duration*22, duration, 10.586972, 21.733656,
40.630966);
rotate(LeftElbow, begin_time+duration*22, duration, -4, -42.449604, 0);
rotate(LeftWrist, begin_time+duration*22, duration, 12.836153, 5.736596,
1.96156);
rotate(RightCollar, begin_time+duration*22, duration, -6.44554, -0.480343, -
0.054265);
rotate(RightShoulder, begin_time+duration*22, duration, -12.071487, 9.831986, -
31.906277);
rotate(RightElbow, begin_time+duration*22, duration, 4, -108.814178, 0);
rotate(RightWrist, begin_time+duration*22, duration, -27.973234, -7.383608,
9.334347);
rotate(Neck, begin_time+duration*22, duration, 3.895567, 18.591354, -1.009541);
rotate(Head, begin_time+duration*22, duration, -40.388779, -8.084021,
37.048977);
/* frame #23 */
move(begin_time+duration*23, duration, 1.209053, 38.622673, 21.208391);
rotate(Hips, begin_time+duration*23, duration, -9.266823, -4.372524, 55.323879);
rotate(LeftHip, begin_time+duration*23, duration, 14.083264, -29.319777,
18.571838);
rotate(LeftKnee, begin_time+duration*23, duration, 5, 33.846798, 0);
rotate(LeftAnkle, begin_time+duration*23, duration, -1.720928, -0.053655, -
14.011722);
rotate(RightHip, begin_time+duration*23, duration, 0.079901, -13.845605, -
6.810368);
rotate(RightKnee, begin_time+duration*23, duration, -5, 39.562054, 0);
rotate(RightAnkle, begin_time+duration*23, duration, 20.816032, -11.384058,
31.552313);
rotate(Chest, begin_time+duration*23, duration, 0.159912, 11.293466, 5.564816);
rotate(LeftCollar, begin_time+duration*23, duration, -5.236822, -0.08023, -
0.007354);
rotate(LeftShoulder, begin_time+duration*23, duration, 12.714739, 9.208652,
41.285271);
rotate(LeftElbow, begin_time+duration*23, duration, -4, -49.757801, 0);
rotate(LeftWrist, begin_time+duration*23, duration, 7.726374, -1.46271,
3.407733);
rotate(RightCollar, begin_time+duration*23, duration, -5.236822, -0.08023, -
0.007354);
rotate(RightShoulder, begin_time+duration*23, duration, -10.581875, -11.200128,
32.006363);
rotate(RightElbow, begin_time+duration*23, duration, 4, -78.285194, 0);
rotate(RightWrist, begin_time+duration*23, duration, -2.273346, -8.601684,
28.718184);
rotate(Neck, begin_time+duration*23, duration, 5.529352, 15.76406, -1.42918);
rotate(Head, begin_time+duration*23, duration, -33.765934, -13.474038,
34.800201);
/* frame #24 */
move(begin_time+duration*24, duration, 5.528447, 40.486179, 20.170446);
rotate(Hips, begin_time+duration*24, duration, -25.422392, -4.919353, 46.13913);
```

```
rotate(LeftHip, begin_time+duration*24, duration, 23.287176, -31.314747,
24.893343);
rotate(LeftKnee, begin_time+duration*24, duration, 5, 28.97089, 0);
rotate(LeftAnkle, begin_time+duration*24, duration, -2.883003, 0.42521, -
10.7356);
rotate(RightHip, begin_time+duration*24, duration, -8.025644, -20.350048, -
14.635918);
rotate(RightKnee, begin_time+duration*24, duration, -5, 107.748329, 0);
rotate(RightAnkle, begin_time+duration*24, duration, 19.199432, -5.196389,
32.009861);
rotate(Chest, begin_time+duration*24, duration, -1.600101, 10.327445, 5.501929);
rotate(LeftCollar, begin_time+duration*24, duration, -6.285758, 1.281315,
0.141124);
rotate(LeftShoulder, begin_time+duration*24, duration, 15.17265, 11.617924,
38.204876);
rotate(LeftElbow, begin_time+duration*24, duration, -4, -64.79818, 0);
rotate(LeftWrist, begin_time+duration*24, duration, 8.811529, -0.867518, -
6.709317);
rotate(RightCollar, begin_time+duration*24, duration, -6.285758, 1.281315,
0.141124);
rotate(RightShoulder, begin_time+duration*24, duration, -12.982543, -0.62157,
13.000268);
rotate(RightElbow, begin_time+duration*24, duration, 4, -78.361313, 0);
rotate(RightWrist, begin_time+duration*24, duration, -3.319278, 0.232943,
23.298136);
rotate(Neck, begin_time+duration*24, duration, 3.955162, 15.633722, -1.022982);
rotate(Head, begin_time+duration*24, duration, -11.66857, -4.483545, 25.844225);
/* frame #25 */
move(begin_time+duration*25, duration, 5.295537, 39.789093, 20.39995);
rotate(Hips, begin_time+duration*25, duration, -23.604849, -1.721839,
43.752861);
rotate(LeftHip, begin_time+duration*25, duration, 21.591957, -33.897446,
23.427464);
rotate(LeftKnee, begin_time+duration*25, duration, 5, 34.914654, 0);
rotate(LeftAnkle, begin_time+duration*25, duration, -1.219767, -2.795517, -
8.374044);
rotate(RightHip, begin_time+duration*25, duration, -7.928692, -16.957268, -
9.393888);
rotate(RightKnee, begin_time+duration*25, duration, -5, 103.923203, 0);
rotate(RightAnkle, begin_time+duration*25, duration, 13.294197, -4.880538,
27.314169);
rotate(Chest, begin_time+duration*25, duration, -3.033443, 13.793843, 6.657612);
rotate(LeftCollar, begin_time+duration*25, duration, -6.061796, 4.566717,
0.484437);
rotate(LeftShoulder, begin_time+duration*25, duration, 12.328124, 4.898986,
29.648855);
rotate(LeftElbow, begin_time+duration*25, duration, -4, -58.688061, 0);
rotate(LeftWrist, begin_time+duration*25, duration, 7.623813, 1.37745,
1.448031);
rotate(RightCollar, begin_time+duration*25, duration, -6.061796, 4.566717,
0.484437);
rotate(RightShoulder, begin_time+duration*25, duration, -5.556199, -34.018394,
13.702803);
rotate(RightElbow, begin_time+duration*25, duration, 4, -79.989006, 0);
rotate(RightWrist, begin_time+duration*25, duration, -37.669003, -3.906067,
9.902341);
rotate(Neck, begin_time+duration*25, duration, -11.446192, 17.360151, 2.947257);
```

```
rotate(Head, begin_time+duration*25, duration, 45.120842, -26.892254, -
35.96677);
/* frame #26 */
move(begin_time+duration*26, duration, 3.889522, 39.692429, 17.946449);
rotate(Hips, begin_time+duration*26, duration, -16.124165, -0.377408,
39.952892);
rotate(LeftHip, begin_time+duration*26, duration, 13.516575, -29.859524,
24.144753);
rotate(LeftKnee, begin_time+duration*26, duration, 5, 27.792677, 0);
rotate(LeftAnkle, begin_time+duration*26, duration, 2.158591, 5.043839, -
9.156518);
rotate(RightHip, begin_time+duration*26, duration, 5.506041, -7.981768, -
5.484976);
rotate(RightKnee, begin_time+duration*26, duration, -5, 36.415352, 0);
rotate(RightAnkle, begin_time+duration*26, duration, 21.782772, -4.26731,
28.310415);
rotate(Chest, begin_time+duration*26, duration, -3.039731, 10.571193, 3.677983);
rotate(LeftCollar, begin_time+duration*26, duration, -4.6826, 3.693718,
0.302337);
rotate(LeftShoulder, begin_time+duration*26, duration, 12.329152, 6.4403,
28.438547);
rotate(LeftElbow, begin_time+duration*26, duration, -4, -68.942276, 0);
rotate(LeftWrist, begin_time+duration*26, duration, 4.385295, -0.769803, -
11.09371);
rotate(RightCollar, begin_time+duration*26, duration, -4.6826, 3.693718,
0.302337);
rotate(RightShoulder, begin_time+duration*26, duration, -15.550063, -16.569849,
39.534958);
rotate(RightElbow, begin_time+duration*26, duration, 4, -75.986748, 0);
rotate(RightWrist, begin_time+duration*26, duration, 18.541346, -11.716288,
26.256393);
rotate(Neck, begin_time+duration*26, duration, -10.055019, 18.185556, 2.594551);
rotate(Head, begin_time+duration*26, duration, 43.277214, -22.843807, -
37.812119);
/* frame #27 */
move(begin_time+duration*27, duration, -2.57177, 36.176712, 15.616888);
rotate(Hips, begin_time+duration*27, duration, -2.308031, -2.7381, 40.438904);
rotate(LeftHip, begin_time+duration*27, duration, 16.770575, -35.345516,
29.3347);
rotate(LeftKnee, begin_time+duration*27, duration, 5, 41.837193, 0);
rotate(LeftAnkle, begin_time+duration*27, duration, -9.611611, 11.4775, -
14.525229);
rotate(RightHip, begin_time+duration*27, duration, 5.632391, -13.537875,
1.116931);
rotate(RightKnee, begin_time+duration*27, duration, -5, 47.788643, 0);
rotate(RightAnkle, begin_time+duration*27, duration, 8.563615, -23.033157,
16.999723);
rotate(Chest, begin_time+duration*27, duration, -1.50815, 22.090057, 4.373504);
rotate(LeftCollar, begin_time+duration*27, duration, -4.555606, 7.951198,
0.631481);
rotate(LeftShoulder, begin_time+duration*27, duration, 10.851443, 9.237918,
28.396049);
rotate(LeftElbow, begin_time+duration*27, duration, -4, -66.368057, 0);
rotate(LeftWrist, begin_time+duration*27, duration, 7.951232, 2.707653, -
4.877942);
rotate(RightCollar, begin_time+duration*27, duration, -4.555606, 7.951198,
0.631481);
```

```
rotate(RightShoulder, begin_time+duration*27, duration, -22.802879, -9.992848, -
10.396385);
rotate(RightElbow, begin_time+duration*27, duration, 4, -95.978516, 0);
rotate(RightWrist, begin_time+duration*27, duration, -4.372321, -0.126252, -
1.783599);
rotate(Neck, begin_time+duration*27, duration, -12.778556, 19.567966, 3.293904);
rotate(Head, begin_time+duration*27, duration, 40.196484, -27.590004, -
38.561165);
/* frame #28 */
move(begin_time+duration*28, duration, -4.822866, 37.171768, 13.764529);
rotate(Hips, begin_time+duration*28, duration, 1.541422, 2.560648, 49.488503);
rotate(LeftHip, begin_time+duration*28, duration, 6.917636, -44.488064,
14.705234);
rotate(LeftKnee, begin_time+duration*28, duration, 5, 76.791924, 0);
rotate(LeftAnkle, begin_time+duration*28, duration, -11.864952, 6.468704, -
21.802341);
rotate(RightHip, begin_time+duration*28, duration, 0.833844, -18.348209, -
2.458627);
rotate(RightKnee, begin_time+duration*28, duration, -5, 44.385151, 0);
rotate(RightAnkle, begin_time+duration*28, duration, 5.988908, -17.960691, -
0.181275);
rotate(Chest, begin_time+duration*28, duration, -4.036631, 25.96159, 5.158026);
rotate(LeftCollar, begin_time+duration*28, duration, -4.826259, 9.955523,
0.836301);
rotate(LeftShoulder, begin_time+duration*28, duration, 11.479845, 6.880339,
28.49959);
rotate(LeftElbow, begin_time+duration*28, duration, -4, -69.853249, 0);
rotate(LeftWrist, begin_time+duration*28, duration, 4.008965, 0.283221, -
11.292403);
rotate(RightCollar, begin_time+duration*28, duration, -4.826259, 9.955523,
0.836301);
rotate(RightShoulder, begin_time+duration*28, duration, -24.907511, -5.331572, -
6.41894);
rotate(RightElbow, begin_time+duration*28, duration, 4, -69.77861, 0);
rotate(RightWrist, begin_time+duration*28, duration, 20.547754, -7.124692,
34.496571);
rotate(Neck, begin_time+duration*28, duration, -14.648149, 19.733112, 3.7682);
rotate(Head, begin_time+duration*28, duration, 41.505211, -32.068718, -
41.735741);
/* frame #29 */
move(begin_time+duration*29, duration, -8.392499, 37.568256, 10.349502);
rotate(Hips, begin_time+duration*29, duration, -0.880307, 2.097676, 56.59845);
rotate(LeftHip, begin_time+duration*29, duration, 6.380307, -30.534266,
10.078238);
rotate(LeftKnee, begin_time+duration*29, duration, 5, 75.598961, 0);
rotate(LeftAnkle, begin_time+duration*29, duration, -16.625631, 3.772563, -
26.726191);
rotate(RightHip, begin_time+duration*29, duration, -0.014247, -26.14138, -
7.148061);
rotate(RightKnee, begin_time+duration*29, duration, -5, 40.418068, 0);
rotate(RightAnkle, begin_time+duration*29, duration, 5.732457, -5.06425, -
3.260262);
rotate(Chest, begin_time+duration*29, duration, 1.657787, 29.260616, 5.385055);
rotate(LeftCollar, begin_time+duration*29, duration, 0.658163, 11.681379, -
0.133263);
rotate(LeftShoulder, begin_time+duration*29, duration, 8.469751, 0.095909,
15.112729);
```

```
rotate(LeftElbow, begin_time+duration*29, duration, -4, -59.393879, 0);
rotate(LeftWrist, begin_time+duration*29, duration, 6.157248, 2.316613, -
5.719873);
rotate(RightCollar, begin_time+duration*29, duration, 0.658163, 11.681379, -
0.133263);
rotate(RightShoulder, begin_time+duration*29, duration, -30.32481, -27.049122,
18.639601);
rotate(RightElbow, begin_time+duration*29, duration, 4, -85.743935, 0);
rotate(RightWrist, begin_time+duration*29, duration, 16.533678, -11.007975,
11.687526);
rotate(Neck, begin_time+duration*29, duration, -16.229736, 20.628698, 4.172568);
rotate(Head, begin_time+duration*29, duration, 44.62019, -48.264965, -
48.327618);
/* frame #30 */
move(begin_time+duration*30, duration, -11.410457, 37.967773, 5.062967);
rotate(Hips, begin_time+duration*30, duration, -1.954735, 1.590503, 57.993626);
rotate(LeftHip, begin_time+duration*30, duration, 0.665692, -11.03483, -
4.201931);
rotate(LeftKnee, begin_time+duration*30, duration, 5, 50.791046, 0);
rotate(LeftAnkle, begin_time+duration*30, duration, -10.653038, -21.671894, -
27.039658);
rotate(RightHip, begin_time+duration*30, duration, -3.020166, -29.032789, -
9.220001);
rotate(RightKnee, begin_time+duration*30, duration, -5, 25.297237, 0);
rotate(RightAnkle, begin_time+duration*30, duration, 7.740826, 16.218435, -
10.746003);
rotate(Chest, begin_time+duration*30, duration, 3.403421, 25.960133, 5.612906);
rotate(LeftCollar, begin_time+duration*30, duration, 0.65632, 10.875768, -
0.12384);
rotate(LeftShoulder, begin_time+duration*30, duration, 8.995813, 2.678878,
12.463011);
rotate(LeftElbow, begin_time+duration*30, duration, -4, -68.871918, 0);
rotate(LeftWrist, begin_time+duration*30, duration, 2.612062, 2.183733, -
11.942036);
rotate(RightCollar, begin_time+duration*30, duration, 0.65632, 10.875768, -
0.12384);
rotate(RightShoulder, begin_time+duration*30, duration, -21.933325, -8.319797, -
29.344229);
rotate(RightElbow, begin_time+duration*30, duration, 4, -63.314133, 0);
rotate(RightWrist, begin_time+duration*30, duration, -28.551807, 5.675143,
1.922075);
rotate(Neck, begin_time+duration*30, duration, -15.236484, 19.383072, 3.91438);
rotate(Head, begin_time+duration*30, duration, 40.647404, -42.486149, -
44.481358);
/* frame #31 */
move(begin_time+duration*31, duration, -13.310488, 37.293152, 2.300948);
rotate(Hips, begin_time+duration*31, duration, -3.139405, -7.635883, 37.455051);
rotate(LeftHip, begin_time+duration*31, duration, 2.633699, -6.630615, -
5.469357);
rotate(LeftKnee, begin_time+duration*31, duration, 5, 43.050732, 0);
rotate(LeftAnkle, begin_time+duration*31, duration, -6.104699, -24.548595,
2.783302);
rotate(RightHip, begin_time+duration*31, duration, 4.761744, -26.933111, -
11.726852);
rotate(RightKnee, begin_time+duration*31, duration, -5, 35.730747, 0);
rotate(RightAnkle, begin_time+duration*31, duration, 9.645547, -5.561864,
2.538472);
```

```
rotate(Chest, begin_time+duration*31, duration, -3.168489, 24.912287, 3.501546);
rotate(LeftCollar, begin_time+duration*31, duration, -1.228648, 10.390729,
0.221632);
rotate(LeftShoulder, begin_time+duration*31, duration, 12.467385, 5.564476,
16.681211);
rotate(LeftElbow, begin_time+duration*31, duration, -4, -63.565086, 0);
rotate(LeftWrist, begin_time+duration*31, duration, 4.508263, 5.421501, -
1.870268);
rotate(RightCollar, begin_time+duration*31, duration, -1.228648, 10.390729,
0.221632);
rotate(RightShoulder, begin_time+duration*31, duration, -20.098883, -17.000519,
22.931446);
rotate(RightElbow, begin_time+duration*31, duration, 4, -91.091461, 0);
rotate(RightWrist, begin_time+duration*31, duration, 33.29866, -8.86493,
30.832458);
rotate(Neck, begin_time+duration*31, duration, -11.212866, 17.514687, 2.888247);
rotate(Head, begin_time+duration*31, duration, 38.199116, -28.742443, -
29.206419);
/* frame #32 */
move(begin_time+duration*32, duration, -15.405882, 37.193916, 1.166987);
rotate(Hips, begin_time+duration*32, duration, -8.331405, -14.428432,
18.479952);
rotate(LeftHip, begin_time+duration*32, duration, 12.665569, -3.405827,
4.004302);
rotate(LeftKnee, begin_time+duration*32, duration, 5, 48.650379, 0);
rotate(LeftAnkle, begin_time+duration*32, duration, -12.401196, -26.108685, -
0.927687);
rotate(RightHip, begin_time+duration*32, duration, 1.693632, -23.095806, -
11.176542);
rotate(RightKnee, begin_time+duration*32, duration, -5, 91.68586, 0);
rotate(RightAnkle, begin_time+duration*32, duration, 10.931064, -4.327225,
12.399897);
rotate(Chest, begin_time+duration*32, duration, -6.785426, 13.805151, -
5.160072);
rotate(LeftCollar, begin_time+duration*32, duration, 1.940723, 4.911746, -
0.16623);
rotate(LeftShoulder, begin_time+duration*32, duration, 15.869948, 9.356386,
31.297867);
rotate(LeftElbow, begin_time+duration*32, duration, -4, -57.093994, 0);
rotate(LeftWrist, begin_time+duration*32, duration, -5.51446, -1.540991, -
12.074597);
rotate(RightCollar, begin_time+duration*32, duration, 1.940723, 4.911746, -
0.16623);
rotate(RightShoulder, begin_time+duration*32, duration, -11.633245, 9.18638, -
42.096085);
rotate(RightElbow, begin_time+duration*32, duration, 4, -76.779984, 0);
rotate(RightWrist, begin_time+duration*32, duration, -56.755745, 12.166543, -
8.44885);
rotate(Neck, begin_time+duration*32, duration, 0.07485, 17.677572, -0.019394);
rotate(Head, begin_time+duration*32, duration, 13.740778, 0.053854, 0.720446);
/* frame #33 */
move(begin_time+duration*33, duration, -19.498203, 36.791943, 0.794329);
rotate(Hips, begin_time+duration*33, duration, -11.924713, -13.101288, -
7.982447);
rotate(LeftHip, begin_time+duration*33, duration, 17.982744, 3.932591,
12.97126);
rotate(LeftKnee, begin_time+duration*33, duration, 5, 47.89658, 0);
```

```
rotate(LeftAnkle, begin_time+duration*33, duration, -18.170898, -23.477152, -
1.364851);
rotate(RightHip, begin_time+duration*33, duration, -0.994282, -10.714174, -
4.775324);
rotate(RightKnee, begin_time+duration*33, duration, -5, 58.432186, 0);
rotate(RightAnkle, begin_time+duration*33, duration, 10.960928, -13.710193,
8.094123);
rotate(Chest, begin_time+duration*33, duration, -8.74569, 5.75772, -13.036322);
rotate(LeftCollar, begin_time+duration*33, duration, 4.592936, -0.963693,
0.077414);
rotate(LeftShoulder, begin_time+duration*33, duration, 59.305523, 20.276937,
66.889015);
rotate(LeftElbow, begin_time+duration*33, duration, -4, -35.73101, 0);
rotate(LeftWrist, begin_time+duration*33, duration, 5.48445, 5.886531,
28.589991);
rotate(RightCollar, begin_time+duration*33, duration, 4.592936, -0.963693,
0.077414);
rotate(RightShoulder, begin_time+duration*33, duration, -13.281172, 28.19022, -
11.781963);
rotate(RightElbow, begin_time+duration*33, duration, 4, -97.033272, 0);
rotate(RightWrist, begin_time+duration*33, duration, -20.566349, -6.905654,
5.214949);
rotate(Neck, begin_time+duration*33, duration, 8.534014, 19.373833, -2.208047);
rotate(Head, begin_time+duration*33, duration, -20.747683, -3.638372,
35.435364);
/* frame #34 */
move(begin_time+duration*34, duration, -25.04064, 34.541718, -2.152383);
rotate(Hips, begin_time+duration*34, duration, -9.507161, -14.3474, -14.12506);
rotate(LeftHip, begin_time+duration*34, duration, 31.241415, -3.894808,
20.217606);
rotate(LeftKnee, begin_time+duration*34, duration, 5, 60.726501, 0);
rotate(LeftAnkle, begin_time+duration*34, duration, -22.793787, -2.983771, -
3.468489);
rotate(RightHip, begin_time+duration*34, duration, -4.420239, -5.518964, -
4.641855);
rotate(RightKnee, begin_time+duration*34, duration, -5, 55.450089, 0);
rotate(RightAnkle, begin_time+duration*34, duration, 20.278519, -21.795588,
7.078641);
rotate(Chest, begin_time+duration*34, duration, -13.331734, 4.25767, -
18.994078);
rotate(LeftCollar, begin_time+duration*34, duration, 10.795991, 0.237421, -
0.045273);
rotate(LeftShoulder, begin_time+duration*34, duration, 57.540062, -0.464859,
52.633423);
rotate(LeftElbow, begin_time+duration*34, duration, -4, -67.522949, 0);
rotate(LeftWrist, begin_time+duration*34, duration, -6.853534, -1.291079, -
25.703667);
rotate(RightCollar, begin_time+duration*34, duration, 10.795991, 0.237421, -
0.045273);
rotate(RightShoulder, begin_time+duration*34, duration, -12.629799, 17.995018, -
8.155565);
rotate(RightElbow, begin_time+duration*34, duration, 4, -98.910812, 0);
rotate(RightWrist, begin_time+duration*34, duration, -23.475235, -24.340931,
5.257693);
rotate(Neck, begin_time+duration*34, duration, 9.298252, 19.731173, -2.405566);
rotate(Head, begin_time+duration*34, duration, -35.009453, -15.438086,
49.158993);
```

```
/* frame #35 */
move(begin_time+duration*35, duration, -25.933212, 37.491116, -4.550582);
rotate(Hips, begin_time+duration*35, duration, 0.064168, -16.884375, -
12.688066);
rotate(LeftHip, begin_time+duration*35, duration, 18.347055, -34.447468,
7.39695);
rotate(LeftKnee, begin_time+duration*35, duration, 5, 58.667515, 0);
rotate(LeftAnkle, begin_time+duration*35, duration, -12.705337, 8.612717, -
11.516697);
rotate(RightHip, begin_time+duration*35, duration, -9.739106, 1.13955, -
3.526858);
rotate(RightKnee, begin_time+duration*35, duration, -5, 38.462833, 0);
rotate(RightAnkle, begin_time+duration*35, duration, 12.335284, -18.818624,
4.008617);
rotate(Chest, begin_time+duration*35, duration, -0.80235, 5.734572, 12.540101);
rotate(LeftCollar, begin_time+duration*35, duration, -12.910357, 3.141355,
0.719666);
rotate(LeftShoulder, begin_time+duration*35, duration, 11.62114, 24.946251,
9.854632);
rotate(LeftElbow, begin_time+duration*35, duration, -4, -83.762062, 0);
rotate(LeftWrist, begin_time+duration*35, duration, -13.300276, -7.354089, -
18.739983);
rotate(RightCollar, begin_time+duration*35, duration, -12.910357, 3.141355,
0.719666);
rotate(RightShoulder, begin_time+duration*35, duration, -48.901688, 0.67588, -
40.840015);
rotate(RightElbow, begin_time+duration*35, duration, 4, -71.720406, 0);
rotate(RightWrist, begin_time+duration*35, duration, -15.253336, -27.368246,
24.660452);
rotate(Neck, begin_time+duration*35, duration, 8.28696, 19.001554, -2.143467);
rotate(Head, begin_time+duration*35, duration, -25.752352, -5.272869, 39.334);
/* frame #36 */
move(begin_time+duration*36, duration, -25.667921, 36.654266, -5.848319);
rotate(Hips, begin_time+duration*36, duration, -1.609781, -23.31484, -8.301706);
rotate(LeftHip, begin_time+duration*36, duration, 13.303544, -43.185051,
6.698026);
rotate(LeftKnee, begin_time+duration*36, duration, 5, 39.34129, 0);
rotate(LeftAnkle, begin_time+duration*36, duration, -13.892073, 1.732412, -
7.595978);
rotate(RightHip, begin_time+duration*36, duration, -8.024762, 3.385848, -
2.933138);
rotate(RightKnee, begin_time+duration*36, duration, -5, 45.428181, 0);
rotate(RightAnkle, begin_time+duration*36, duration, 13.308956, -19.975626, -
0.442079);
rotate(Chest, begin_time+duration*36, duration, 7.971937, 15.905271, 17.210348);
rotate(LeftCollar, begin_time+duration*36, duration, -9.448284, 3.894366,
0.647554);
rotate(LeftShoulder, begin_time+duration*36, duration, 10.212002, 34.165905,
32.198166);
rotate(LeftElbow, begin_time+duration*36, duration, -4.000002, -90.358276, 0);
rotate(LeftWrist, begin_time+duration*36, duration, -0.677769, -8.432377,
4.00422);
rotate(RightCollar, begin_time+duration*36, duration, -9.448284, 3.894366,
0.647554);
rotate(RightShoulder, begin_time+duration*36, duration, -72.810669, -8.118636, -
44.113529);
rotate(RightElbow, begin_time+duration*36, duration, 4, -78.542618, 0);
```

```
rotate(RightWrist, begin_time+duration*36, duration, -13.880897, -8.315669,
5.097155);
rotate(Neck, begin_time+duration*36, duration, 6.985496, 19.233885, -1.808986);
rotate(Head, begin_time+duration*36, duration, -18.00268, -0.148159, 29.662563);
/* frame #37 */
move(begin_time+duration*37, duration, -28.251032, 37.343399, -5.765088);
rotate(Hips, begin_time+duration*37, duration, -2.023721, -20.531796, -
27.444828);
rotate(LeftHip, begin_time+duration*37, duration, 18.116037, -24.457521,
4.153266);
rotate(LeftKnee, begin_time+duration*37, duration, 5, 45.396858, 0);
rotate(LeftAnkle, begin_time+duration*37, duration, -14.365676, 1.221082, -
5.933489);
rotate(RightHip, begin_time+duration*37, duration, -9.43992, -2.850604, -
4.487241);
rotate(RightKnee, begin_time+duration*37, duration, -5, 44.812656, 0);
rotate(RightAnkle, begin_time+duration*37, duration, 6.812588, -20.922262,
9.448046);
rotate(Chest, begin_time+duration*37, duration, -0.559373, 12.179784, -2.59826);
rotate(LeftCollar, begin_time+duration*37, duration, 5.339542, 5.214218, -
0.486654);
rotate(LeftShoulder, begin_time+duration*37, duration, 50.331554, 16.836998,
70.910561);
rotate(LeftElbow, begin_time+duration*37, duration, -4, -50.157284, 0);
rotate(LeftWrist, begin_time+duration*37, duration, 26.888718, -11.650026, -
15.048403);
rotate(RightCollar, begin_time+duration*37, duration, 5.339542, 5.214218, -
0.486654);
rotate(RightShoulder, begin_time+duration*37, duration, -11.036438, -27.667927,
43.137444);
rotate(RightElbow, begin_time+duration*37, duration, 4, -82.158752, 0);
rotate(RightWrist, begin_time+duration*37, duration, 0.031549, -15.778079,
11.354571);
rotate(Neck, begin_time+duration*37, duration, 6.374015, 22.001444, -1.659223);
rotate(Head, begin_time+duration*37, duration, -39.174412, -6.650928,
46.544353);
/* frame #38 */
move(begin_time+duration*38, duration, -27.940968, 37.820374, -3.458864);
rotate(Hips, begin_time+duration*38, duration, -4.833589, -14.69658, -
30.384657);
rotate(LeftHip, begin_time+duration*38, duration, 12.988768, -3.791139,
3.738446);
rotate(LeftKnee, begin_time+duration*38, duration, 5, 28.512779, 0);
rotate(LeftAnkle, begin_time+duration*38, duration, -7.349823, 2.403576, -
5.498552);
rotate(RightHip, begin_time+duration*38, duration, -6.08619, -0.207666, -
3.85537);
rotate(RightKnee, begin_time+duration*38, duration, -5, 36.08947, 0);
rotate(RightAnkle, begin_time+duration*38, duration, 6.597594, -17.62145,
8.943601);
rotate(Chest, begin_time+duration*38, duration, 0.080207, 6.298342, -6.925882);
rotate(LeftCollar, begin_time+duration*38, duration, 8.23331, 3.508477, -
0.507331);
rotate(LeftShoulder, begin_time+duration*38, duration, 42.067711, -8.374337, -
1.464297);
rotate(LeftElbow, begin_time+duration*38, duration, -4, -25.083193, 0);
```

```
rotate(LeftWrist, begin_time+duration*38, duration, 1.287169, -20.171288, -
57.009834);
rotate(RightCollar, begin_time+duration*38, duration, 8.23331, 3.508477, -
0.507331);
rotate(RightShoulder, begin_time+duration*38, duration, -20.412849, 30.116459, -
4.143721);
rotate(RightElbow, begin_time+duration*38, duration, 4, -84.859512, 0);
rotate(RightWrist, begin_time+duration*38, duration, -12.803655, -11.334308, -
4.026666);
rotate(Neck, begin_time+duration*38, duration, 7.072407, 20.75647, -1.835742);
rotate(Head, begin_time+duration*38, duration, -41.706367, -14.227448,
48.718861);
/* frame #39 */
move(begin_time+duration*39, duration, -26.473856, 37.693752, 0.626075);
rotate(Hips, begin_time+duration*39, duration, -9.527833, -9.702545, -
28.566137);
rotate(LeftHip, begin_time+duration*39, duration, 16.621702, -3.700997,
7.897642);
rotate(LeftKnee, begin_time+duration*39, duration, 5, 35.792042, 0);
rotate(LeftAnkle, begin_time+duration*39, duration, -21.953972, -0.391614, -
7.981704);
rotate(RightHip, begin_time+duration*39, duration, -3.570017, -1.36263, -
4.006276);
rotate(RightKnee, begin_time+duration*39, duration, -5, 44.335072, 0);
rotate(RightAnkle, begin_time+duration*39, duration, 11.197388, -20.755516,
9.977957);
rotate(Chest, begin_time+duration*39, duration, -2.561728, 9.361875, -
10.081392);
rotate(LeftCollar, begin_time+duration*39, duration, 6.853405, 2.239757, -
0.269122);
rotate(LeftShoulder, begin_time+duration*39, duration, 12.04427, 9.447991, -
17.278099);
rotate(LeftElbow, begin_time+duration*39, duration, -4, -5.929834, 0);
rotate(LeftWrist, begin_time+duration*39, duration, -19.483088, 6.812008, -
52.702965);
rotate(RightCollar, begin_time+duration*39, duration, 6.853405, 2.239757, -
0.269122);
rotate(RightShoulder, begin_time+duration*39, duration, -23.340425, 6.265081, -
68.24279);
rotate(RightElbow, begin_time+duration*39, duration, 4, -100.639374, 0);
rotate(RightWrist, begin_time+duration*39, duration, -39.627609, -0.220975, -
11.124305);
rotate(Neck, begin_time+duration*39, duration, 10.066321, 19.855564, -2.603054);
rotate(Head, begin_time+duration*39, duration, -40.819401, -23.374737,
50.518841);
/* frame #40 */
move(begin_time+duration*40, duration, -21.295252, 38.855175, 2.802833);
rotate(Hips, begin_time+duration*40, duration, -4.113865, -0.800927, -5.731671);
rotate(LeftHip, begin_time+duration*40, duration, 5.207735, -16.179394,
11.599999);
rotate(LeftKnee, begin_time+duration*40, duration, 5, 29.112108, 0);
rotate(LeftAnkle, begin_time+duration*40, duration, -6.028343, 0.045828, -
13.349515);
rotate(RightHip, begin_time+duration*40, duration, -7.45044, -0.546817, -
3.763365);
rotate(RightKnee, begin_time+duration*40, duration, -5, 32.744991, 0);
```

```
rotate(RightAnkle, begin_time+duration*40, duration, 15.106, -12.94623, -
4.89036);
rotate(Chest, begin_time+duration*40, duration, -0.641912, 5.653935, 5.455912);
rotate(LeftCollar, begin_time+duration*40, duration, -7.575138, -1.277816, -
0.169918);
rotate(LeftShoulder, begin_time+duration*40, duration, 11.709727, 17.971699,
10.85696);
rotate(LeftElbow, begin_time+duration*40, duration, -4, -9.649094, 0);
rotate(LeftWrist, begin_time+duration*40, duration, -5.787847, 6.193623, -
35.397987);
rotate(RightCollar, begin_time+duration*40, duration, -7.575138, -1.277816, -
0.169918);
rotate(RightShoulder, begin_time+duration*40, duration, -41.926865, -24.121065,
-35.988075);
rotate(RightElbow, begin_time+duration*40, duration, 4, -111.460632, 0);
rotate(RightWrist, begin_time+duration*40, duration, -28.773535, 2.307665, -
6.109928);
rotate(Neck, begin_time+duration*40, duration, 2.238824, 18.76194, -0.580571);
rotate(Head, begin_time+duration*40, duration, -9.561075, 0.593498, 15.504861);
/* frame #41 */
move(begin_time+duration*41, duration, -20.117479, 36.488071, 2.007215);
rotate(Hips, begin_time+duration*41, duration, -1.055172, 3.791689, 32.832253);
rotate(LeftHip, begin_time+duration*41, duration, -7.82034, -31.674412,
5.349044);
rotate(LeftKnee, begin_time+duration*41, duration, 5, 42.037693, 0);
rotate(LeftAnkle, begin_time+duration*41, duration, 0.76563, -8.035349, -
28.748308);
rotate(RightHip, begin_time+duration*41, duration, -7.74136, -6.230703, -
4.948108);
rotate(RightKnee, begin_time+duration*41, duration, -5, 59.13047, 0);
rotate(RightAnkle, begin_time+duration*41, duration, 15.00759, -18.700087, -
2.992904);
rotate(Chest, begin_time+duration*41, duration, 7.912317, 26.459326, 19.890676);
rotate(LeftCollar, begin_time+duration*41, duration, -4.009331, 10.046657,
0.700537);
rotate(LeftShoulder, begin_time+duration*41, duration, 7.774262, -7.993758, -
7.745814);
rotate(LeftElbow, begin_time+duration*41, duration, -4, -27.428101, 0);
rotate(LeftWrist, begin_time+duration*41, duration, -7.913921, 8.199841, -
32.486355);
rotate(RightCollar, begin_time+duration*41, duration, -4.009331, 10.046657,
0.700537);
rotate(RightShoulder, begin_time+duration*41, duration, -32.562931, -38.401741,
-3.129745);
rotate(RightElbow, begin_time+duration*41, duration, 4, -4.578099, 0);
rotate(RightWrist, begin_time+duration*41, duration, -17.876163, 11.243464,
5.870867);
rotate(Neck, begin_time+duration*41, duration, -19.608232, 22.12426, 5.030983);
rotate(Head, begin_time+duration*41, duration, 39.726376, -51.551128, -
53.448574);
/* frame #42 */
move(begin_time+duration*42, duration, -19.921675, 35.640514, 3.830442);
rotate(Hips, begin_time+duration*42, duration, -3.318364, 8.926889, 50.412998);
rotate(LeftHip, begin_time+duration*42, duration, -12.212711, -36.157623,
4.545377);
rotate(LeftKnee, begin_time+duration*42, duration, 5, 47.115849, 0);
```

```
rotate(LeftAnkle, begin_time+duration*42, duration, 2.413134, -13.702175, -
36.224197);
rotate(RightHip, begin_time+duration*42, duration, -4.343373, -9.105464, -
4.437965);
rotate(RightKnee, begin_time+duration*42, duration, -5, 67.167229, 0);
rotate(RightAnkle, begin_time+duration*42, duration, 15.06088, -13.234879,
2.215832);
rotate(Chest, begin_time+duration*42, duration, 11.510859, 30.992752,
20.214298);
rotate(LeftCollar, begin_time+duration*42, duration, 1.230595, 10.874016, -
0.232186);
rotate(LeftShoulder, begin_time+duration*42, duration, 5.624515, -2.465557, -
6.571714);
rotate(LeftElbow, begin_time+duration*42, duration, -4, -22.849693, 0);
rotate(LeftWrist, begin_time+duration*42, duration, -9.082065, 7.902856, -
32.974533);
rotate(RightCollar, begin_time+duration*42, duration, 1.230595, 10.874016, -
0.232186);
rotate(RightShoulder, begin_time+duration*42, duration, -36.086315, -39.812031,
-25.76437);
rotate(RightElbow, begin_time+duration*42, duration, 4, -4.359483, 0);
rotate(RightWrist, begin_time+duration*42, duration, -13.550258, 4.749774,
16.813965);
rotate(Neck, begin_time+duration*42, duration, -23.940771, 23.749792, 6.120804);
rotate(Head, begin_time+duration*42, duration, 47.886288, -59.913128, -
53.452007);
/* frame #43 */
move(begin_time+duration*43, duration, -17.504612, 36.300442, 5.217095);
rotate(Hips, begin_time+duration*43, duration, -5.628019, 5.479823, 41.682171);
rotate(LeftHip, begin_time+duration*43, duration, -8.601374, -30.591553,
2.907157);
rotate(LeftKnee, begin_time+duration*43, duration, 5, 46.756134, 0);
rotate(LeftAnkle, begin_time+duration*43, duration, 1.457158, -16.886118, -
21.684658);
rotate(RightHip, begin_time+duration*43, duration, -1.977467, -8.790771, -
1.451337);
rotate(RightKnee, begin_time+duration*43, duration, -5, 61.62175, 0);
rotate(RightAnkle, begin_time+duration*43, duration, 14.561487, 6.699501,
6.124833);
rotate(Chest, begin_time+duration*43, duration, 6.639979, 27.662107, 20.004814);
rotate(LeftCollar, begin_time+duration*43, duration, -1.311238, 10.551497,
0.240153);
rotate(LeftShoulder, begin_time+duration*43, duration, 5.614146, -6.337554, -
9.302183);
rotate(LeftElbow, begin_time+duration*43, duration, -4, -25.914902, 0);
rotate(LeftWrist, begin_time+duration*43, duration, -8.807321, 5.79468, -
32.118752);
rotate(RightCollar, begin_time+duration*43, duration, -1.311238, 10.551497,
0.240153);
rotate(RightShoulder, begin_time+duration*43, duration, -26.130737, -44.356796,
8.490204);
rotate(RightElbow, begin_time+duration*43, duration, 4, -14.259245, 0);
rotate(RightWrist, begin_time+duration*43, duration, -10.934694, 0.021061,
22.855698);
rotate(Neck, begin_time+duration*43, duration, -21.624071, 21.484648, 5.520024);
rotate(Head, begin_time+duration*43, duration, 42.802063, -61.519215, -
52.948505);
```

```
/* frame #44 */
move(begin_time+duration*44, duration, -15.6111, 38.41444, 7.120049);
rotate(Hips, begin_time+duration*44, duration, -8.062617, -1.248211, 23.590809);
rotate(LeftHip, begin_time+duration*44, duration, -3.136358, -12.857987,
0.260752);
rotate(LeftKnee, begin_time+duration*44, duration, 5, 31.212231, 0);
rotate(LeftAnkle, begin_time+duration*44, duration, -2.073405, -12.502839, -
7.97001);
rotate(RightHip, begin_time+duration*44, duration, -2.466807, -8.390903,
1.258919);
rotate(RightKnee, begin_time+duration*44, duration, -5, 46.90276, 0);
rotate(RightAnkle, begin_time+duration*44, duration, 7.3846, 37.362823,
14.124306);
rotate(Chest, begin_time+duration*44, duration, 1.276495, 14.677749, 10.278434);
rotate(LeftCollar, begin_time+duration*44, duration, -2.667545, 4.66781,
0.217237);
rotate(LeftShoulder, begin_time+duration*44, duration, 10.828405, -5.138576, -
8.845365);
rotate(LeftElbow, begin_time+duration*44, duration, -4, -21.917793, 0);
rotate(LeftWrist, begin_time+duration*44, duration, -12.572578, 3.870594, -
33.784359);
rotate(RightCollar, begin_time+duration*44, duration, -2.667545, 4.66781,
0.217237);
rotate(RightShoulder, begin_time+duration*44, duration, -8.238424, -29.943775,
36.65062);
rotate(RightElbow, begin_time+duration*44, duration, 4, -23.49864, 0);
rotate(RightWrist, begin_time+duration*44, duration, 5.10302, -2.814797,
42.915646);
rotate(Neck, begin_time+duration*44, duration, -9.463994, 15.70164, 2.439373);
rotate(Head, begin_time+duration*44, duration, 28.068354, -24.952259, -
24.54398);
/* frame #45 */
move(begin_time+duration*45, duration, -14.983106, 39.670475, 9.361086);
rotate(Hips, begin_time+duration*45, duration, -10.44239, -2.025074, 9.484221);
rotate(LeftHip, begin_time+duration*45, duration, -1.416184, 3.624989, -
1.644559);
rotate(LeftKnee, begin_time+duration*45, duration, 5, 9.125307, 0);
rotate(LeftAnkle, begin_time+duration*45, duration, -0.238154, -2.478077, -
2.258921);
rotate(RightHip, begin_time+duration*45, duration, -1.336528, -6.543489,
1.598327);
rotate(RightKnee, begin_time+duration*45, duration, -5, 24.900906, 0);
rotate(RightAnkle, begin_time+duration*45, duration, 12.377325, 1.977337,
9.391264);
rotate(Chest, begin_time+duration*45, duration, -1.12273, 6.540425, -0.120186);
rotate(LeftCollar, begin_time+duration*45, duration, -0.564021, 0.805625,
0.007931);
rotate(LeftShoulder, begin_time+duration*45, duration, 14.581067, -3.669213,
6.43038);
rotate(LeftElbow, begin_time+duration*45, duration, -4, -23.132156, 0);
rotate(LeftWrist, begin_time+duration*45, duration, -6.773915, 4.505913, -
19.661072);
rotate(RightCollar, begin_time+duration*45, duration, -0.564021, 0.805625,
0.007931);
rotate(RightShoulder, begin_time+duration*45, duration, -11.967134, -11.053391,
8.926049);
rotate(RightElbow, begin_time+duration*45, duration, 4, -22.224499, 0);
```

```
rotate(RightWrist, begin_time+duration*45, duration, -2.018004, -4.315001,
34.008465);
rotate(Neck, begin_time+duration*45, duration, -0.262461, 15.251215, 0.06793);
rotate(Head, begin_time+duration*45, duration, 2.956564, -9.484528, 3.682611);
/* frame #46 */
move(begin_time+duration*46, duration, -17.159521, 39.607914, 11.100108);
rotate(Hips, begin_time+duration*46, duration, -8.204349, -2.42259, 3.901974);
rotate(LeftHip, begin_time+duration*46, duration, -0.337786, 8.187469, -
1.994571);
rotate(LeftKnee, begin_time+duration*46, duration, 5, 7.681812, 0);
rotate(LeftAnkle, begin_time+duration*46, duration, -5.002226, -2.480394, -
0.174493);
rotate(RightHip, begin_time+duration*46, duration, 2.58057, -2.877165,
2.137016);
rotate(RightKnee, begin_time+duration*46, duration, -5, 10.606837, 0);
rotate(RightAnkle, begin_time+duration*46, duration, 4.610003, 3.547747,
2.267411);
rotate(Chest, begin_time+duration*46, duration, -1.045276, 2.270353, -0.117146);
rotate(LeftCollar, begin_time+duration*46, duration, 0.644537, 0.241684, -
0.002719);
rotate(LeftShoulder, begin_time+duration*46, duration, 9.867409, 0.891811,
6.673345);
rotate(LeftElbow, begin_time+duration*46, duration, -4, -18.052942, 0);
rotate(LeftWrist, begin_time+duration*46, duration, -8.666321, 3.902874, -
20.755356);
rotate(RightCollar, begin_time+duration*46, duration, 0.644537, 0.241684, -
0.002719);
rotate(RightShoulder, begin_time+duration*46, duration, -14.099945, 0.375251,
6.016263);
rotate(RightElbow, begin_time+duration*46, duration, 4, -16.370586, 0);
rotate(RightWrist, begin_time+duration*46, duration, -1.883581, -0.32088,
28.547237);
rotate(Neck, begin_time+duration*46, duration, 1.117766, 16.324501, -0.28936);
rotate(Head, begin_time+duration*46, duration, -7.249373, -4.82253, 11.246789);
/* frame #47 */
move(begin_time+duration*47, duration, -22.067722, 39.080452, 12.936417);
rotate(Hips, begin_time+duration*47, duration, -9.676706, -3.234099, 4.819626);
rotate(LeftHip, begin_time+duration*47, duration, 9.907595, 1.448668, -
1.871075);
rotate(LeftKnee, begin_time+duration*47, duration, 5, 26.042086, 0);
rotate(LeftAnkle, begin_time+duration*47, duration, -14.86352, -7.832563, -
2.852821);
rotate(RightHip, begin_time+duration*47, duration, 7.724608, 2.846064,
2.023222);
rotate(RightKnee, begin_time+duration*47, duration, -5, 6.026187, 0);
rotate(RightAnkle, begin_time+duration*47, duration, 1.400003, 2.368321, -
4.292152);
rotate(Chest, begin_time+duration*47, duration, 0.724116, 1.222816, -1.586017);
rotate(LeftCollar, begin_time+duration*47, duration, 4.028991, 1.044775, -
0.073585);
rotate(LeftShoulder, begin_time+duration*47, duration, 5.674289, 2.178767,
0.364512);
rotate(LeftElbow, begin_time+duration*47, duration, -4, -13.448117, 0);
rotate(LeftWrist, begin_time+duration*47, duration, -17.657927, 5.49244, -
34.710487);
rotate(RightCollar, begin_time+duration*47, duration, 4.028991, 1.044775, -
0.073585);
```

```
rotate(RightShoulder, begin_time+duration*47, duration, -9.718236, 5.151939,
1.341975);
rotate(RightElbow, begin_time+duration*47, duration, 4, -12.859472, 0);
rotate(RightWrist, begin_time+duration*47, duration, 1.641112, 0.649004,
32.924183);
rotate(Neck, begin_time+duration*47, duration, 2.250007, 16.450851, -0.582395);
rotate(Head, begin_time+duration*47, duration, -8.57336, -2.486886, 14.352758);
/* frame #48 */
move(begin_time+duration*48, duration, -22.951071, 38.718277, 12.844002);
rotate(Hips, begin_time+duration*48, duration, -6.142839, -3.762676, 6.185045);
rotate(LeftHip, begin_time+duration*48, duration, 7.176509, -3.54602, -
0.398779);
rotate(LeftKnee, begin_time+duration*48, duration, 5, 34.585258, 0);
rotate(LeftAnkle, begin_time+duration*48, duration, -13.713072, -6.581944,
4.561676);
rotate(RightHip, begin_time+duration*48, duration, 5.553437, 2.410469,
2.866866);
rotate(RightKnee, begin_time+duration*48, duration, -5, 6.403542, 0);
rotate(RightAnkle, begin_time+duration*48, duration, 1.230407, 2.206066, -
6.066314);
rotate(Chest, begin_time+duration*48, duration, 3.537684, 2.354404, -1.487586);
rotate(LeftCollar, begin_time+duration*48, duration, 2.255895, 0.322016, -
0.012685);
rotate(LeftShoulder, begin_time+duration*48, duration, 5.806876, 1.010114, -
1.635041);
rotate(LeftElbow, begin_time+duration*48, duration, -4, -22.331606, 0);
rotate(LeftWrist, begin_time+duration*48, duration, -15.657457, 2.614702, -
30.282511);
rotate(RightCollar, begin_time+duration*48, duration, 2.255895, 0.322016, -
0.012685);
rotate(RightShoulder, begin_time+duration*48, duration, -13.299748, 1.666905,
1.247501);
rotate(RightElbow, begin_time+duration*48, duration, 4, -16.534197, 0);
rotate(RightWrist, begin_time+duration*48, duration, 4.46215, -1.076906,
40.865871);
rotate(Neck, begin_time+duration*48, duration, 0.436139, 17.078217, -0.112954);
rotate(Head, begin_time+duration*48, duration, -8.163432, 0.312578, 10.589978);
```

```
/*****************************
 * dance.bvh
 *****************************/

HIERARCHY
ROOT Hips
{
	OFFSET	0.000000	0.000000	0.000000
	CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
	JOINT LeftHip
	{
		OFFSET	5.500000	0.000000	0.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftKnee
		{
			OFFSET	0.000000	-17.500000	0.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftAnkle
			{
				OFFSET	0.000000	-17.000000	0.000000
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET	0.000000	-4.700000	-2.000000
				}
			}
		}
	}
	JOINT RightHip
	{
		OFFSET	-5.500000	0.000000	0.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT RightKnee
		{
			OFFSET	0.000000	-17.500000	0.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT RightAnkle
			{
				OFFSET	0.000000	-17.000000	0.000000
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET	0.000000	-4.700000	-2.000000
				}
			}
		}
	}
	JOINT Chest
	{
		OFFSET	0.000000	3.000000	-1.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftCollar
		{
			OFFSET	0.000000	18.000000	1.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftShoulder
			{
```

```
                  OFFSET      6.500000    0.000000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT LeftElbow
                  {
                        OFFSET      0.000000    -11.000000  0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT LeftWrist
                        {
                              OFFSET      0.000000    -10.500000  0.000000
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              End Site
                              {
                                    OFFSET      0.000000    -6.000000
0.000000
                              }
                        }
                  }
            }
      }
      JOINT RightCollar
      {
            OFFSET      0.000000    18.000000   1.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightShoulder
            {
                  OFFSET      -6.500000   0.000000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT RightElbow
                  {
                        OFFSET      0.000000    -11.000000  0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT RightWrist
                        {
                              OFFSET      0.000000    -10.500000  0.000000
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              End Site
                              {
                                    OFFSET      0.000000    -6.000000
0.000000
                              }
                        }
                  }
            }
      }
      JOINT Neck
      {
            OFFSET      0.000000    18.000000   1.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT Head
            {
                  OFFSET      0.000000    7.750000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  End Site
                  {
                        OFFSET      0.000000    7.000000    0.000000
                  }
            }
```

```
            }
      }
}
MOTION
Frames: 50
Frame Time: 0.2
-17.129814  39.513103   -20.553329  -5.145142   7.454098    -35.545376
      18.684629   -12.699254  28.043344   5.000000    14.012280   0.000000    -
8.197182    -8.943927   5.615734    15.744060   0.171168    0.018856    -
5.000000    3.507612    0.000000    -4.224754   1.468446    -6.937200   5.154825
      -0.541715   2.757841    -0.483571   1.530707    0.012918    27.021511
      11.128203   23.415245   -4.000000   -21.375710  0.000000    -22.103354
      2.792870    -33.793991  -0.483571   1.530707    0.012918    -31.893370
      12.959333   57.074883   4.000000    -80.039268  0.000000    4.898635    -
38.379738   58.616867   6.651874    18.752653   -1.721906   -31.482822  -
9.030337    35.319271
-16.701021  39.610031   -20.517260  -5.724495   7.937147    -35.742962
      18.322882   -12.275520  28.749619   5.000000    11.085456   0.000000    -
5.985360    -3.260179   5.566837    15.824615   0.172553    0.340880    -
5.000000    4.094175    0.000000    -4.022447   0.901529    -6.397221   5.395926
      -0.379808   2.282838    0.644868    1.852910    -0.020852   14.775752
      7.906818    14.836170   -4.000000   -22.547981  0.000000    -29.504467
      4.250135    -34.309799  0.644868    1.852910    -0.020852   -27.681364
      10.861183   55.300331   4.000000    -78.663719  0.000000    0.312259    -
42.018234   56.770298   6.407799    18.573019   -1.658621   -31.783743  -
9.183754    34.867287
-16.436958  39.467709   -20.750069  -5.682179   8.576076    -36.695980
      19.263140   -13.699969  29.574968   5.000000    11.926646   0.000000    -
5.939101    0.649740    5.577283    15.670573   -0.224643   0.906330    -
5.000000    3.155029    0.000000    -3.106385   1.047569    -6.190315   5.230749
      0.908727    2.196691    0.322572    2.497519    -0.014057   10.100171
      5.033052    6.380805    -4.000000   -23.067116  0.000000    -27.581787
      4.015338    -39.103401  0.322572    2.497519    -0.014057   -26.958368
      9.854655    54.975403   4.000000    -77.508209  0.000000    -2.196213   -
44.679024   56.582664   6.304765    18.206894   -1.631415   -32.519417  -
10.743377   34.826012
-17.444820  39.173840   -19.847218  -4.301334   10.141496   -36.106544
      21.454668   -16.970825  31.112751   5.000000    18.638451   0.000000    -
6.733837    -2.014010   5.583443    13.825427   -0.655887   0.967337    -
5.000000    2.266317    0.000000    -2.755840   0.106372    -2.390693   3.055874
      2.031169    1.959889    0.080670    2.900388    -0.004082   15.606045
      2.967969    4.837171    -4.000000   -27.886517  0.000000    -31.164696  -
0.430840    -40.531345  0.080670    2.900388    -0.004082   -28.718487
      10.454666   56.434525   4.000000    -77.251305  0.000000    -1.579567   -
44.004166   58.816631   6.814987    17.901215   -1.762387   -32.143200  -
13.020011   34.309849
-13.804417  39.363987   -16.844353  -3.278627   6.159461    -20.895088  8.001950
      -19.956074  21.958853   5.000000    19.896765   0.000000    -1.895744
      3.342982    -4.131546   4.755366    1.913827    -1.040583   -5.000000
      11.558072   0.000000    6.703470    -9.655918   -3.271017   7.798990
      2.369120    2.125154    -2.497621   0.402449    0.017554    23.418095   -
3.267248    32.554077   -4.000000   -33.269577  0.000000    -0.384012   8.584818
      6.856423    -2.497621   0.402449    0.017554    -29.005827  17.245050
      34.535248   4.000000    -70.186607  0.000000    -10.889338  -40.835850
      50.585796   2.191028    17.524885   -0.567506   -21.379515  -4.004979
      20.406166
```

```
-12.913258  36.754375   -15.254353  6.202761    -10.001901  0.365909    -
9.308453    -14.229591  6.657677    5.000000    37.161026   0.000000    -
2.449677    -9.497665   -3.009280   -13.667568  -14.627858  -9.706951   -
5.000000    75.428650   0.000000    8.848930    1.987457    0.395745    3.273613
      13.472042   9.141914    -5.735439   4.168554    0.418305    6.883256    -
4.370080    -2.815387   -4.000000   -72.103004  0.000000    -5.323997   8.678768
      -2.287296   -5.735439   4.168554    0.418305    -9.934323   -10.115631
      0.270280    4.000000    -40.831001  0.000000    -30.275377  -3.769179
      8.136178    -3.990515   16.393581   1.032363    2.644384    -4.640108   -
6.501391
-13.729616  38.915710   -14.079136  -4.687915   -2.721347   3.006787    1.750564
      -14.481229  4.332572    5.000000    25.709202   0.000000    -1.944163   -
3.521772    -9.299974   -8.281845   -22.865160  -10.094368  -5.000000
      50.943615   0.000000    11.056540   -1.060217   4.828407    2.310683
      16.612484   4.596787    -2.424496   4.507697    0.190662    6.613963    -
9.679655    -16.146868  -4.000000   -74.046700  0.000000    2.458456    0.441579
      -12.047024  -2.424496   4.507697    0.190662    -4.260518   -17.980421
      29.872219   4.000000    -72.128159  0.000000    -8.674229   -16.570129
      45.654068   -0.966407   14.966805   0.250113    4.394180    -12.313986  -
1.102398
-17.605885  38.095810   -11.617993  -5.271516   -1.365486   -3.223443
      10.560674   -15.811641  6.759228    5.000000    37.040127   0.000000    -
9.443357    -10.716975  -10.119300  -14.779498  -20.627630  -12.051445  -
5.000000    13.870152   0.000000    5.313439    18.437410   -4.815678   2.705829
      18.385557   6.465448    -3.969739   6.028090    0.417547    11.212226   -
3.132380    0.631595    -4.000000   -60.894497  0.000000    3.739194    9.478383
      -3.233555   -3.969739   6.028090    0.417547    -7.502120   -18.328396
      28.859022   4.000000    -59.123852  0.000000    -7.490026   -14.430984
      47.442421   -1.079134   13.942091   0.279332    3.657167    -16.902508
      0.938171
-23.600647  37.698734   -10.166685  -2.185839   0.385279    1.215283
      17.258251   -12.420403  7.362471    5.000000    26.835659   0.000000    -
17.740488   0.045887    -15.781089  -16.552633  -25.895638  -13.750904  -
5.000000    41.909149   0.000000    16.062405   4.098430    7.506119    1.593486
      16.692188   4.087021    -3.068497   3.861659    0.206853    13.304035
      3.396105    4.504075    -4.000000   -61.148354  0.000000    -0.880527
      10.686496   -5.476295   -3.068497   3.861659    0.206853    -8.718978   -
8.713498    23.496515   4.000000    -63.392948  0.000000    -13.404161  -
5.152517    30.957378   -0.359833   13.944358   0.093147    -2.638163   -
15.480479   5.779716
-31.304668  37.535427   -10.720301  6.427833    -4.060543   0.601263
      19.330179   -28.018993  10.517758   5.000000    60.249142   0.000000    -
18.865295   8.206586    -23.602928  -12.014886  -18.610247  -9.646481   -
5.000001    42.457005   0.000000    15.281750   -11.541622  14.299189   -
1.905475    21.768213   3.794538    -5.253003   4.492848    0.412642    8.180956
      1.949255    13.262990   -4.000000   -68.251106  0.000000    4.578137
      5.445683    -0.036052   -5.253003   4.492848    0.412642    -11.833835  -
2.295370    21.839031   4.000000    -62.177822  0.000000    -13.533296  -
3.750202    33.125626   -0.676574   14.967844   0.175106    -8.830094   -
16.107822   4.833903
-32.732391  38.542297   -11.074305  6.351815    -7.001144   -6.431266   6.705881
      -52.716469  14.462562   5.000000    86.088631   0.000000    -25.022865
      0.534009    -27.579309  -7.907940   -14.431717  -7.414193   -5.000000
      33.901268   0.000000    8.193705    -5.243793   16.185028   2.369667
      28.536413   6.702434    -5.131217   8.426060    0.753865    7.422952
      0.441218    13.327086   -4.000000   -75.195961  0.000000    12.174783   -
5.499401    -18.963175  -5.131217   8.426060    0.753865    -15.204421  -
```

```
5.032105    2.209987    4.000000    -58.643513  0.000000    -16.825235  -
2.586789    27.067694   -0.363799   14.090647   0.094169    -20.503429  -
20.385145   8.081726
-32.278660  39.272934   -11.011002  6.154453    -15.451610  -20.755604  0.893926
      -59.990433  10.573915   5.000000    85.813202   0.000000    -25.480740  -
2.385731    -23.452936  -8.574259   -8.349257   -5.190028   -5.000000
      29.429222   0.000000    -0.697423   -1.757561   9.854114    -0.632914
      26.520632   9.159431    -9.220221   9.067887    1.465505    10.180879
      6.381096    16.545647   -4.000000   -59.051235  0.000000    12.747101
      9.152156    -5.138736   -9.220221   9.067887    1.465505    -21.508057  -
16.779799   -6.544375   4.000000    -92.664253  0.000000    -21.055349  -
12.317038   26.088436   3.489380    15.035939   -0.902598   -29.706583  -
24.043907   22.460142
-33.316418  36.989594   -6.949919   -3.084077   -23.768581  -27.830767  5.669065
      -32.096844  8.338368    5.000000    89.195702   0.000000    -21.951626  -
4.681513    -22.342213  -7.449198   -0.625773   -3.457067   -5.000000
      49.150726   0.000000    4.168019    -22.077457  13.293082   5.330661
      18.794889   10.027803   -8.074035   3.749193    0.531461    7.730994
      6.135199    20.182251   -4.000000   -61.816410  0.000000    9.851315
      4.738596    -7.813559   -8.074035   3.749193    0.531461    -22.609959  -
31.443661   -3.000330   4.000000    -76.438675  0.000000    -19.632473  -
6.424749    16.513079   3.844430    18.518120   -0.996227   -29.897303  -
4.928665    30.755003
-35.012768  38.488434   -10.228033  5.212686    -9.265230   -10.699771  2.858577
      -42.437805  7.272922    5.000000    66.438011   0.000000    -19.621138  -
5.615705    -19.896460  -0.838473   -8.305875   -3.088899   -5.000000
      30.726648   0.000000    4.394121    -14.292957  -2.139857   -1.352536
      18.786697   5.105765    -3.709799   2.572730    0.166756    6.869457    -
0.884872    19.218727   -4.000000   -65.588669  0.000000    9.487166    3.076603
      -12.155961  -3.709799   2.572730    0.166756    -12.527830  4.982889
      8.816996    4.000000    -69.731430  0.000000    -17.809412  -11.324987
      15.585416   2.449895    16.113960   -0.634022   -23.467354  -9.043563
      23.815760
-33.663204  36.736336   -7.270027   -1.695279   -9.798649   -9.286841
      11.939426   -31.224306  10.923769   5.000000    48.903454   0.000000    -
15.902702   -6.770611   -9.040105   1.939302    -9.087251   -2.718635   -
5.000000    44.696926   0.000000    6.471078    -20.643593  7.620894    -
1.269280    22.895517   2.448728    -1.462410   7.409721    0.188638    8.210942
      6.011745    24.847458   -4.000000   -58.536957  0.000000    14.590768
      6.098452    -1.076313   -1.462410   7.409721    0.188638    -11.519151
      2.730728    -5.626769   4.000000    -75.558502  0.000000    -19.310085  -
18.617662   18.274012   4.756541    15.521576   -1.229825   -27.818129  -
23.093639   26.447117
-31.593407  38.429569   -0.107606   -7.236069   2.736379    -4.116490
      18.453445   -24.334566  21.183193   5.000000    29.462717   0.000000    -
7.232381    1.510060    -8.790439   3.067137    1.642617    -2.357990   -
5.000000    24.662708   0.000000    13.708294   -19.177113  5.760456    2.979552
      -0.224835   -0.701676   2.258437    -2.737133   0.107905    8.297648
      9.200754    24.730993   -4.000000   -49.402401  0.000000    11.464435
      3.255547    -5.322170   2.258437    -2.737133   0.107905    -17.064674  -
16.852499   -37.412354  4.000000    -103.862350 0.000000    -47.784214  -
1.877796    4.783830    2.700413    18.682549   -0.700135   -20.926788  0.008929
      26.309801
-25.697178  38.038990   3.508533    -8.597879   -9.990374   10.366679   7.568030
      -16.418871  8.130315    5.000000    41.720665   0.000000    -4.262243   -
10.700898   -5.033810   -2.819038   -24.394457  -7.705383   -5.000000
      91.214020   0.000000    -1.649781   11.472948   19.753998   0.560178
```

```
      9.923924    -4.169472   3.224058    -1.689218   0.095140    13.134015
      8.216235    24.048523   -4.000000   -58.366673  0.000000    5.170369
      0.339265    -6.074597   3.224058    -1.689218   0.095140    -35.296440
      9.963168    -28.391800  4.000000    -84.823120  0.000000    -5.706198   -
10.867316   3.552155    2.239107    16.654568   -0.579631   -9.830771   -
2.520955    19.868963
-22.647726  39.401943   6.762849    -9.141970   -21.029650  29.310394   8.839565
      0.531282    0.545652    5.000000    32.148582   0.000000    1.425214    -
14.855791   -5.115057   5.361639    -51.340919  -5.630099   -5.000000
      108.192879  0.000000    6.257185    -6.955666   18.854788   4.315859
      11.943705   -9.674056   8.218926    -0.956869   0.138203    11.437817
      9.199423    24.217394   -4.000000   -55.829403  0.000000    7.666867
      4.597013    -5.731146   8.218926    -0.956869   0.138203    -23.990576  -
11.461184   10.906994   4.000000    -98.185417  0.000000    22.987438   1.622884
      18.847523   0.712060    18.921223   -0.184723   5.011434    5.271060    -
0.919087
-19.576277  38.724869   9.098734    -4.021911   -13.947841  42.548473   -
0.105155    4.482029    -5.653007   5.000000    25.560541   0.000000    1.923022
      -19.407925  -1.312229   11.252844   -44.133507  -1.537978   -5.000000
      49.356453   0.000000    17.826344   -6.808752   8.581303    2.482679
      8.717219    -8.613018   6.123101    -0.240320   0.025781    9.799450
      21.014687   27.550228   -4.000000   -52.686272  0.000000    3.784122
      4.010980    -6.963547   6.123101    -0.240320   0.025781    -17.884668
      18.353355   -60.230564  4.000001    -91.421753  0.000000    -34.974564
      7.662100    0.714494    -0.998303   17.426891   0.258600    15.330964
      1.089054    -8.786468
-12.350068  36.049206   18.580812   -3.908316   -5.693626   50.656548   -
0.756879    7.715332    -7.548730   5.000000    41.097740   0.000000    -
9.326496    -26.550417  -12.502378  14.993808   -31.326401  -0.211357   -
5.000000    35.164063   0.000000    2.206248    6.756851    9.054711    -
0.963585    4.445596    -15.663724  6.848312    -0.399958   0.048034
      14.801471   14.147343   30.496218   -4.000000   -56.911797  0.000000
      4.816602    1.049923    -0.127078   6.848312    -0.399958   0.048034    -
22.998663   12.359883   0.093860    4.000000    -93.555344  0.000000
      10.581180   -5.635666   24.960760   -2.953969   16.921141   0.764666
      13.503547   -5.643531   -14.019279
-7.114614   38.435638   19.081347   1.343216    -1.574224   56.730480
      16.064003   -29.910963  19.326490   5.000000    91.905159   0.000000    -
13.328533   -13.095946  -21.585583  1.675439    -16.817625  -6.558056   -
5.000000    26.209188   0.000000    5.651016    2.306720    10.968217   1.603339
      7.104790    -5.341287   -0.725511   -1.933439   -0.024479   12.872150
      17.298964   30.875441   -4.000000   -54.136639  0.000000    5.947587
      4.429058    -2.786671   -0.725511   -1.933439   -0.024479   -17.213139
      19.578344   -57.210838  4.000000    -105.109604 0.000000    -38.383766  -
3.984361    4.766713    -6.399650   17.059870   1.654302    16.146664   -
6.942715    -19.591543
-3.568674   36.438126   19.670063   7.446728    -2.377560   61.941776
      22.193066   -36.092335  34.811600   5.000000    81.862816   0.000000    -
18.798332   -13.096813  -22.381237  -6.312239   -13.925522  -8.365907   -
5.000000    47.359772   0.000000    15.846600   -18.616632  20.380396   4.708108
      14.361793   2.104641    -1.210133   2.980293    0.062927    11.786359
      17.675713   36.412388   -4.000000   -59.467270  0.000000    3.032277    -
0.846374    -3.904777   -1.210133   2.980293    0.062927    -13.366383
      19.284698   -18.468691  4.000000    -76.635895  0.000000    -8.960938
      10.941560   -1.469352   -8.036469   14.953121   2.073633    -0.888819   -
8.378451    -10.934739
```

```
-0.406935   38.298141   21.386162   -3.718359   -1.211248   55.594067   9.228383
      -22.645313  24.767464   5.000000    25.112000   0.000000    -5.572019   -
4.743712    -9.686931   0.789116    -8.213847   -6.313642   -5.000000
      24.619989   0.000000    12.725786   -7.951370   20.842644   6.339537
      5.504952    4.170451    -6.445540   -0.480343   -0.054265   10.586972
      21.733656   40.630966   -4.000000   -42.449604  0.000000    12.836153
      5.736596    1.961560    -6.445540   -0.480343   -0.054265   -12.071487
      9.831986    -31.906277  4.000000    -108.814178 0.000000    -27.973234  -
7.383608    9.334347    3.895567    18.591354   -1.009541   -40.388779  -
8.084021    37.048977
1.209053    38.622673   21.208391   -9.266823   -4.372524   55.323879
      14.083264   -29.319777  18.571838   5.000000    33.846798   0.000000    -
1.720928    -0.053655   -14.011722  0.079901    -13.845605  -6.810368   -
5.000000    39.562054   0.000000    20.816032   -11.384058  31.552313   0.159912
      11.293466   5.564816    -5.236822   -0.080230   -0.007354   12.714739
      9.208652    41.285271   -4.000000   -49.757801  0.000000    7.726374    -
1.462710    3.407733    -5.236822   -0.080230   -0.007354   -10.581875  -
11.200128   32.006363   4.000000    -78.285194  0.000000    -2.273346   -
8.601684    28.718184   5.529352    15.764060   -1.429180   -33.765934  -
13.474038   34.800201
5.528447    40.486179   20.170446   -25.422392  -4.919353   46.139130
      23.287176   -31.314747  24.893343   5.000000    28.970890   0.000000    -
2.883003    0.425210    -10.735600  -8.025644   -20.350048  -14.635918  -
5.000000    107.748329  0.000000    19.199432   -5.196389   32.009861   -
1.600101    10.327445   5.501929    -6.285758   1.281315    0.141124
      15.172650   11.617924   38.204876   -4.000000   -64.798180  0.000000
      8.811529    -0.867518   -6.709317   -6.285758   1.281315    0.141124    -
12.982543   -0.621570   13.000268   4.000000    -78.361313  0.000000    -
3.319278    0.232943    23.298136   3.955162    15.633722   -1.022982   -
11.668570   -4.483545   25.844225
5.295537    39.789093   20.399950   -23.604849  -1.721839   43.752861
      21.591957   -33.897446  23.427464   5.000000    34.914654   0.000000    -
1.219767    -2.795517   -8.374044   -7.928692   -16.957268  -9.393888   -
5.000000    103.923203  0.000000    13.294197   -4.880538   27.314169   -
3.033443    13.793843   6.657612    -6.061796   4.566717    0.484437
      12.328124   4.898986    29.648855   -4.000000   -58.688061  0.000000
      7.623813    1.377450    1.448031    -6.061796   4.566717    0.484437    -
5.556199    -34.018394  13.702803   4.000000    -79.989006  0.000000    -
37.669003   -3.906067   9.902341    -11.446192  17.360151   2.947257
      45.120842   -26.892254  -35.966770
3.889522    39.692429   17.946449   -16.124165  -0.377408   39.952892
      13.516575   -29.859524  24.144753   5.000000    27.792677   0.000000
      2.158591    5.043839    -9.156518   5.506041    -7.981768   -5.484976   -
5.000000    36.415352   0.000000    21.782772   -4.267310   28.310415   -
3.039731    10.571193   3.677983    -4.682600   3.693718    0.302337
      12.329152   6.440300    28.438547   -4.000000   -68.942276  0.000000
      4.385295    -0.769803   -11.093710  -4.682600   3.693718    0.302337    -
15.550063   -16.569849  39.534958   4.000000    -75.986748  0.000000
      18.541346   -11.716288  26.256393   -10.055019  18.185556   2.594551
      43.277214   -22.843807  -37.812119
-2.571770   36.176712   15.616888   -2.308031   -2.738100   40.438904
      16.770575   -35.345516  29.334700   5.000000    41.837193   0.000000    -
9.611611    11.477500   -14.525229  5.632391    -13.537875  1.116931    -
5.000000    47.788643   0.000000    8.563615    -23.033157  16.999723   -
1.508150    22.090057   4.373504    -4.555606   7.951198    0.631481
      10.851443   9.237918    28.396049   -4.000000   -66.368057  0.000000
      7.951232    2.707653    -4.877942   -4.555606   7.951198    0.631481    -
```

```
22.802879   -9.992848   -10.396385  4.000000    -95.978516  0.000000    -
4.372321    -0.126252   -1.783599   -12.778556  19.567966   3.293904
      40.196484   -27.590004  -38.561165
-4.822866   37.171768   13.764529   1.541422    2.560648    49.488503   6.917636
      -44.488064  14.705234   5.000000    76.791924   0.000000    -11.864952
      6.468704    -21.802341  0.833844    -18.348209  -2.458627   -5.000000
      44.385151   0.000000    5.988908    -17.960691  -0.181275   -4.036631
      25.961590   5.158026    -4.826259   9.955523    0.836301    11.479845
      6.880339    28.499590   -4.000000   -69.853249  0.000000    4.008965
      0.283221    -11.292403  -4.826259   9.955523    0.836301    -24.907511  -
5.331572    -6.418940   4.000000    -69.778610  0.000000    20.547754   -
7.124692    34.496571   -14.648149  19.733112   3.768200    41.505211   -
32.068718   -41.735741
-8.392499   37.568256   10.349502   -0.880307   2.097676    56.598450   6.380307
      -30.534266  10.078238   5.000000    75.598961   0.000000    -16.625631
      3.772563    -26.726191  -0.014247   -26.141380  -7.148061   -5.000000
      40.418068   0.000000    5.732457    -5.064250   -3.260262   1.657787
      29.260616   5.385055    0.658163    11.681379   -0.133263   8.469751
      0.095909    15.112729   -4.000000   -59.393879  0.000000    6.157248
      2.316613    -5.719873   0.658163    11.681379   -0.133263   -30.324810  -
27.049122   18.639601   4.000000    -85.743935  0.000000    16.533678   -
11.007975   11.687526   -16.229736  20.628698   4.172568    44.620190   -
48.264965   -48.327618
-11.410457  37.967773   5.062967    -1.954735   1.590503    57.993626   0.665692
      -11.034830  -4.201931   5.000000    50.791046   0.000000    -10.653038  -
21.671894   -27.039658  -3.020166   -29.032789  -9.220001   -5.000000
      25.297237   0.000000    7.740826    16.218435   -10.746003  3.403421
      25.960133   5.612906    0.656320    10.875768   -0.123840   8.995813
      2.678878    12.463011   -4.000000   -68.871918  0.000000    2.612062
      2.183733    -11.942036  0.656320    10.875768   -0.123840   -21.933325  -
8.319797    -29.344229  4.000000    -63.314133  0.000000    -28.551807  5.675143
      1.922075    -15.236484  19.383072   3.914380    40.647404   -42.486149  -
44.481358
-13.310488  37.293152   2.300948    -3.139405   -7.635883   37.455051   2.633699
      -6.630615   -5.469357   5.000000    43.050732   0.000000    -6.104699   -
24.548595   2.783302    4.761744    -26.933111  -11.726852  -5.000000
      35.730747   0.000000    9.645547    -5.561864   2.538472    -3.168489
      24.912287   3.501546    -1.228648   10.390729   0.221632    12.467385
      5.564476    16.681211   -4.000000   -63.565086  0.000000    4.508263
      5.421501    -1.870268   -1.228648   10.390729   0.221632    -20.098883  -
17.000519   22.931446   4.000000    -91.091461  0.000000    33.298660   -
8.864930    30.832458   -11.212866  17.514687   2.888247    38.199116   -
28.742443   -29.206419
-15.405882  37.193916   1.166987    -8.331405   -14.428432  18.479952
      12.665569   -3.405827   4.004302    5.000000    48.650379   0.000000    -
12.401196   -26.108685  -0.927687   1.693632    -23.095806  -11.176542  -
5.000000    91.685860   0.000000    10.931064   -4.327225   12.399897   -
6.785426    13.805151   -5.160072   1.940723    4.911746    -0.166230
      15.869948   9.356386    31.297867   -4.000000   -57.093994  0.000000    -
5.514460    -1.540991   -12.074597  1.940723    4.911746    -0.166230   -
11.633245   9.186380    -42.096085  4.000000    -76.779984  0.000000    -
56.755745   12.166543   -8.448850   0.074850    17.677572   -0.019394
      13.740778   0.053854    0.720446
-19.498203  36.791943   0.794329    -11.924713  -13.101288  -7.982447
      17.982744   3.932591    12.971260   5.000000    47.896580   0.000000    -
18.170898   -23.477152  -1.364851   -0.994282   -10.714174  -4.775324   -
5.000000    58.432186   0.000000    10.960928   -13.710193  8.094123    -
```

```
8.745690    5.757720    -13.036322  4.592936    -0.963693   0.077414
      59.305523   20.276937   66.889015   -4.000000   -35.731010  0.000000
      5.484450    5.886531    28.589991   4.592936    -0.963693   0.077414    -
13.281172   28.190220   -11.781963  4.000000    -97.033272  0.000000    -
20.566349   -6.905654   5.214949    8.534014    19.373833   -2.208047   -
20.747683   -3.638372   35.435364
-25.040640  34.541718   -2.152383   -9.507161   -14.347400  -14.125060
      31.241415   -3.894808   20.217606   5.000000    60.726501   0.000000    -
22.793787   -2.983771   -3.468489   -4.420239   -5.518964   -4.641855   -
5.000000    55.450089   0.000000    20.278519   -21.795588  7.078641    -
13.331734   4.257670    -18.994078  10.795991   0.237421    -0.045273
      57.540062   -0.464859   52.633423   -4.000000   -67.522949  0.000000    -
6.853534    -1.291079   -25.703667  10.795991   0.237421    -0.045273   -
12.629799   17.995018   -8.155565   4.000000    -98.910812  0.000000    -
23.475235   -24.340931  5.257693    9.298252    19.731173   -2.405566   -
35.009453   -15.438086  49.158993
-25.933212  37.491116   -4.550582   0.064168    -16.884375  -12.688066
      18.347055   -34.447468  7.396950    5.000000    58.667515   0.000000    -
12.705337   8.612717    -11.516697  -9.739106   1.139550    -3.526858   -
5.000000    38.462833   0.000000    12.335284   -18.818624  4.008617    -
0.802350    5.734572    12.540101   -12.910357  3.141355    0.719666
      11.621140   24.946251   9.854632    -4.000000   -83.762062  0.000000    -
13.300276   -7.354089   -18.739983  -12.910357  3.141355    0.719666    -
48.901688   0.675880    -40.840015  4.000000    -71.720406  0.000000    -
15.253336   -27.368246  24.660452   8.286960    19.001554   -2.143467   -
25.752352   -5.272869   39.334000
-25.667921  36.654266   -5.848319   -1.609781   -23.314840  -8.301706
      13.303544   -43.185051  6.698026    5.000000    39.341290   0.000000    -
13.892073   1.732412    -7.595978   -8.024762   3.385848    -2.933138   -
5.000000    45.428181   0.000000    13.308956   -19.975626  -0.442079   7.971937
      15.905271   17.210348   -9.448284   3.894366    0.647554    10.212002
      34.165905   32.198166   -4.000002   -90.358276  0.000000    -0.677769   -
8.432377    4.004220    -9.448284   3.894366    0.647554    -72.810669  -
8.118636    -44.113529  4.000000    -78.542618  0.000000    -13.880897  -
8.315669    5.097155    6.985496    19.233885   -1.808986   -18.002680  -
0.148159    29.662563
-28.251032  37.343399   -5.765088   -2.023721   -20.531796  -27.444828
      18.116037   -24.457521  4.153266    5.000000    45.396858   0.000000    -
14.365676   1.221082    -5.933489   -9.439920   -2.850604   -4.487241   -
5.000000    44.812656   0.000000    6.812588    -20.922262  9.448046    -
0.559373    12.179784   -2.598260   5.339542    5.214218    -0.486654
      50.331554   16.836998   70.910561   -4.000000   -50.157284  0.000000
      26.888718   -11.650026  -15.048403  5.339542    5.214218    -0.486654   -
11.036438   -27.667927  43.137444   4.000000    -82.158752  0.000000    0.031549
      -15.778079  11.354571   6.374015    22.001444   -1.659223   -39.174412  -
6.650928    46.544353
-27.940968  37.820374   -3.458864   -4.833589   -14.696580  -30.384657
      12.988768   -3.791139   3.738446    5.000000    28.512779   0.000000    -
7.349823    2.403576    -5.498552   -6.086190   -0.207666   -3.855370   -
5.000000    36.089470   0.000000    6.597594    -17.621450  8.943601    0.080207
      6.298342    -6.925882   8.233310    3.508477    -0.507331   42.067711   -
8.374337    -1.464297   -4.000000   -25.083193  0.000000    1.287169    -
20.171288   -57.009834  8.233310    3.508477    -0.507331   -20.412849
      30.116459   -4.143721   4.000000    -84.859512  0.000000    -12.803655  -
11.334308   -4.026666   7.072407    20.756470   -1.835742   -41.706367  -
14.227448   48.718861
```

```
-26.473856  37.693752   0.626075    -9.527833   -9.702545   -28.566137
      16.621702   -3.700997   7.897642    5.000000    35.792042   0.000000    -
21.953972   -0.391614   -7.981704   -3.570017   -1.362630   -4.006276   -
5.000000    44.335072   0.000000    11.197388   -20.755516  9.977957    -
2.561728    9.361875    -10.081392  6.853405    2.239757    -0.269122
      12.044270   9.447991    -17.278099  -4.000000   -5.929834   0.000000    -
19.483088   6.812008    -52.702965  6.853405    2.239757    -0.269122   -
23.340425   6.265081    -68.242790  4.000000    -100.639374 0.000000    -
39.627609   -0.220975   -11.124305  10.066321   19.855564   -2.603054   -
40.819401   -23.374737  50.518841
-21.295252  38.855175   2.802833    -4.113865   -0.800927   -5.731671   5.207735
      -16.179394  11.599999   5.000000    29.112108   0.000000    -6.028343
      0.045828    -13.349515  -7.450440   -0.546817   -3.763365   -5.000000
      32.744991   0.000000    15.106000   -12.946230  -4.890360   -0.641912
      5.653935    5.455912    -7.575138   -1.277816   -0.169918   11.709727
      17.971699   10.856960   -4.000000   -9.649094   0.000000    -5.787847
      6.193623    -35.397987  -7.575138   -1.277816   -0.169918   -41.926865  -
24.121065   -35.988075  4.000000    -111.460632 0.000000    -28.773535  2.307665
      -6.109928   2.238824    18.761940   -0.580571   -9.561075   0.593498
      15.504861
-20.117479  36.488071   2.007215    -1.055172   3.791689    32.832253   -
7.820340    -31.674412  5.349044    5.000000    42.037693   0.000000    0.765630
      -8.035349   -28.748308  -7.741360   -6.230703   -4.948108   -5.000000
      59.130470   0.000000    15.007590   -18.700087  -2.992904   7.912317
      26.459326   19.890676   -4.009331   10.046657   0.700537    7.774262    -
7.993758    -7.745814   -4.000000   -27.428101  0.000000    -7.913921   8.199841
      -32.486355  -4.009331   10.046657   0.700537    -32.562931  -38.401741  -
3.129745    4.000000    -4.578099   0.000000    -17.876163  11.243464   5.870867
      -19.608232  22.124260   5.030983    39.726376   -51.551128  -53.448574
-19.921675  35.640514   3.830442    -3.318364   8.926889    50.412998   -
12.212711   -36.157623  4.545377    5.000000    47.115849   0.000000    2.413134
      -13.702175  -36.224197  -4.343373   -9.105464   -4.437965   -5.000000
      67.167229   0.000000    15.060880   -13.234879  2.215832    11.510859
      30.992752   20.214298   1.230595    10.874016   -0.232186   5.624515    -
2.465557    -6.571714   -4.000000   -22.849693  0.000000    -9.082065   7.902856
      -32.974533  1.230595    10.874016   -0.232186   -36.086315  -39.812031  -
25.764370   4.000000    -4.359483   0.000000    -13.550258  4.749774
      16.813965   -23.940771  23.749792   6.120804    47.886288   -59.913128  -
53.452007
-17.504612  36.300442   5.217095    -5.628019   5.479823    41.682171   -
8.601374    -30.591553  2.907157    5.000000    46.756134   0.000000    1.457158
      -16.886118  -21.684658  -1.977467   -8.790771   -1.451337   -5.000000
      61.621750   0.000000    14.561487   6.699501    6.124833    6.639979
      27.662107   20.004814   -1.311238   10.551497   0.240153    5.614146    -
6.337554    -9.302183   -4.000000   -25.914902  0.000000    -8.807321   5.794680
      -32.118752  -1.311238   10.551497   0.240153    -26.130737  -44.356796
      8.490204    4.000000    -14.259245  0.000000    -10.934694  0.021061
      22.855698   -21.624071  21.484648   5.520024    42.802063   -61.519215  -
52.948505
-15.611100  38.414440   7.120049    -8.062617   -1.248211   23.590809   -
3.136358    -12.857987  0.260752    5.000000    31.212231   0.000000    -
2.073405    -12.502839  -7.970010   -2.466807   -8.390903   1.258919    -
5.000000    46.902760   0.000000    7.384600    37.362823   14.124306   1.276495
      14.677749   10.278434   -2.667545   4.667810    0.217237    10.828405   -
5.138576    -8.845365   -4.000000   -21.917793  0.000000    -12.572578  3.870594
      -33.784359  -2.667545   4.667810    0.217237    -8.238424   -29.943775
      36.650620   4.000000    -23.498640  0.000000    5.103020    -2.814797
```

```
      42.915646   -9.463994   15.701640   2.439373    28.068354   -24.952259  -
24.543980
-14.983106  39.670475   9.361086    -10.442390  -2.025074   9.484221    -
1.416184    3.624989    -1.644559   5.000000    9.125307    0.000000    -
0.238154    -2.478077   -2.258921   -1.336528   -6.543489   1.598327    -
5.000000    24.900906   0.000000    12.377325   1.977337    9.391264    -
1.122730    6.540425    -0.120186   -0.564021   0.805625    0.007931
      14.581067   -3.669213   6.430380    -4.000000   -23.132156  0.000000    -
6.773915    4.505913    -19.661072  -0.564021   0.805625    0.007931    -
11.967134   -11.053391  8.926049    4.000000    -22.224499  0.000000    -
2.018004    -4.315001   34.008465   -0.262461   15.251215   0.067930    2.956564
      -9.484528   3.682611
-17.159521  39.607914   11.100108   -8.204349   -2.422590   3.901974    -
0.337786    8.187469    -1.994571   5.000000    7.681812    0.000000    -
5.002226    -2.480394   -0.174493   2.580570    -2.877165   2.137016    -
5.000000    10.606837   0.000000    4.610003    3.547747    2.267411    -
1.045276    2.270353    -0.117146   0.644537    0.241684    -0.002719   9.867409
      0.891811    6.673345    -4.000000   -18.052942  0.000000    -8.666321
      3.902874    -20.755356  0.644537    0.241684    -0.002719   -14.099945
      0.375251    6.016263    4.000000    -16.370586  0.000000    -1.883581   -
0.320880    28.547237   1.117766    16.324501   -0.289360   -7.249373   -
4.822530    11.246789
-22.067722  39.080452   12.936417   -9.676706   -3.234099   4.819626    9.907595
      1.448668    -1.871075   5.000000    26.042086   0.000000    -14.863520  -
7.832563    -2.852821   7.724608    2.846064    2.023222    -5.000000   6.026187
      0.000000    1.400003    2.368321    -4.292152   0.724116    1.222816    -
1.586017    4.028991    1.044775    -0.073585   5.674289    2.178767    0.364512
      -4.000000   -13.448117  0.000000    -17.657927  5.492440    -34.710487
      4.028991    1.044775    -0.073585   -9.718236   5.151939    1.341975
      4.000000    -12.859472  0.000000    1.641112    0.649004    32.924183
      2.250007    16.450851   -0.582395   -8.573360   -2.486886   14.352758
-22.951071  38.718277   12.844002   -6.142839   -3.762676   6.185045    7.176509
      -3.546020   -0.398779   5.000000    34.585258   0.000000    -13.713072  -
6.581944    4.561676    5.553437    2.410469    2.866866    -5.000000   6.403542
      0.000000    1.230407    2.206066    -6.066314   3.537684    2.354404    -
1.487586    2.255895    0.322016    -0.012685   5.806876    1.010114    -
1.635041    -4.000000   -22.331606  0.000000    -15.657457  2.614702    -
30.282511   2.255895    0.322016    -0.012685   -13.299748  1.666905    1.247501
      4.000000    -16.534197  0.000000    4.462150    -1.076906   40.865871
      0.436139    17.078217   -0.112954   -8.163432   0.312578    10.589978
```

```
/************************
 * dance.sbvh
 ***********************/

HIERARCHY
ROOT Hips
{
	OFFSET	0.000000	0.000000	0.000000
	CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
	JOINT LeftHip
	{
		OFFSET	5.500000	0.000000	0.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftKnee
		{
			OFFSET	0.000000	-17.500000	0.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftAnkle
			{
				OFFSET	0.000000	-17.000000	0.000000
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET	0.000000	-4.700000	-2.000000
				}
			}
		}
	}
	JOINT RightHip
	{
		OFFSET	-5.500000	0.000000	0.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT RightKnee
		{
			OFFSET	0.000000	-17.500000	0.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT RightAnkle
			{
				OFFSET	0.000000	-17.000000	0.000000
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET	0.000000	-4.700000	-2.000000
				}
			}
		}
	}
	JOINT Chest
	{
		OFFSET	0.000000	3.000000	-1.000000
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftCollar
		{
			OFFSET	0.000000	18.000000	1.000000
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftShoulder
			{
```

```
                  OFFSET      6.500000    0.000000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT LeftElbow
                  {
                        OFFSET      0.000000    -11.000000  0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT LeftWrist
                        {
                              OFFSET      0.000000    -10.500000  0.000000
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              End Site
                              {
                                    OFFSET      0.000000    -6.000000
0.000000
                              }
                        }
                  }
            }
      }
      JOINT RightCollar
      {
            OFFSET      0.000000    18.000000   1.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightShoulder
            {
                  OFFSET      -6.500000   0.000000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT RightElbow
                  {
                        OFFSET      0.000000    -11.000000  0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT RightWrist
                        {
                              OFFSET      0.000000    -10.500000  0.000000
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              End Site
                              {
                                    OFFSET      0.000000    -6.000000
0.000000
                              }
                        }
                  }
            }
      }
      JOINT Neck
      {
            OFFSET      0.000000    18.000000   1.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT Head
            {
                  OFFSET      0.000000    7.750000    0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  End Site
                  {
                        OFFSET      0.000000    7.000000    0.000000
                  }
            }
```

```
            }
      }
}
MOTION
Frames: 0
Frame Time: 0.01
```

```
/*
 * Sample ANIMO program intended for testing
 * The result of this program is a BVH file for a marching figure
 *  exuivalent to march.bvh
 * (requires walkfunc.animo and march.sbvh)
 * Author: Alexei Masterov
 */

include "march.sbvh";
include "walkfunc.animo";
include "wavefunc.animo";

//
================================================================================
======
// MAIN:
//
================================================================================
======

num_steps = 25;
step_time = 1;

// move the figure along the Z axis
move ( 0, 1, -0.47, 39.85, -0.12 );
move ( 1, num_steps * step_time, -0.47, 39.85, 150 );

// move legs and arms
for ( i = 0 to num_steps )
{
    if ( i % 2 == 0 )
       left_step( i * step_time, step_time );
    else
       right_step( i * step_time, step_time );
}

wave_right_hand(15 * step_time, step_time / 3);
```

```
/***************************
 * march.bvh
 ***************************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 5.5000 0.0000 -0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET -0.0000 -17.5000 -0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -17.0000 -0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
            OFFSET 0.0000 -4.7000 -2.0000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -5.5000 -0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -17.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
            OFFSET 0.0000 -4.7000 -2.0000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 3.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 18.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 6.5000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -11.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -10.5000 -0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
              OFFSET -0.0000 -6.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 18.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -6.5000 -0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -11.0000 -0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -10.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
              OFFSET 0.0000 -6.0000 -0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 18.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 3.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
          OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames:     12
Frame Time: .2
-0.47 39.85 -0.12 0     0     0     -0.11 -10.12      4.22  0     28.79 5.19  -
0.75  -8.44 -8.67 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     29.83 -33.3 -46.9 0     -66.72      -11.11      -
31.45 10.67 26.4  0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0
-0.47 39.85 2.88  0     0     0     -1.01 -28.21      -1.56 0     71.64 4.87  -
16.91 -17.59      -1.84 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     32.48 -51.08      -62.65      179.97      -
89.22 174.15      -14.81      9.37  17.29 0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0
-0.47 39.85 5.88  0     0     0     4.62  -22.28      3.55  0.01  70.77 5.58  -
24.07 -10.11      -7.22 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     28.23 -55.95      -80.9 0.02  -77.32
      5.77  -10.55      -4.56 -17.88      0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0
-0.47 39.85 8.88  0     0     0     0     0     0     0     0     0     0     0
      0     -5.5  -8.03 -2.54 0     39.65 -5.88 3.44  -18.18      10.08 0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     -36.38      -5.37 24.88 0     -22.49      0.9   30.19 -
7.37  22.78 0     0     0     0     0     0
-0.47 39.85 11.88 0     0     0     0     0     0     0     0     0     0     0
      0     -2.69 -17.24      -0.31 0     57.12 -3.86 8.74  -17.5 4.45  0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     -31.19      -23.73      47.08 0.01  -79.93      5.61
      23.47 21.63 -13.6 0     0     0     0     0     0
-0.47 39.85 14.88 0     0     0     0     0     0     0     0     0     0     0
      0     -0.71 -28.86      -2.08 0     71.07 -3.65 13.63 -8.64 2.02  0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     -28.02      -40.89      58.39 -0.03 -72.54      -3.64
      16.12 7.85  7.15  0     0     0     0     0     0
-0.47 39.85 17.88 0     0     0     -0.11 -10.12      4.22  0     28.79 5.19  -
0.75  -8.44 -8.67 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     29.83 -33.3 -46.9 0     -66.72      -11.11      -
31.45 10.67 26.4  0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0
-0.47 39.85 20.88 0     0     0     -1.01 -28.21      -1.56 0     71.64 4.87  -
16.91 -17.59      -1.84 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     32.48 -51.08      -62.65      179.97      -
89.22 174.15      -14.81      9.37  17.29 0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0
-0.47 39.85 23.88 0     0     0     4.62  -22.28      3.55  0.01  70.77 5.58  -
24.07 -10.11      -7.22 0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     28.23 -55.95      -80.9 0.02  -77.32
      5.77  -10.55      -4.56 -17.88      0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0
-0.47 39.85 26.88 0     0     0     0     0     0     0     0     0     0     0
      0     -5.5  -8.03 -2.54 0     39.65 -5.88 3.44  -18.18      10.08 0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     -36.38      -5.37 24.88 0     -22.49      0.9   30.19 -
7.37  22.78 0     0     0     0     0     0
-0.47 39.85 29.88 0     0     0     0     0     0     0     0     0     0     0
      0     -2.69 -17.24      -0.31 0     57.12 -3.86 8.74  -17.5 4.45  0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
      0     0     0     -31.19      -23.73      47.08 0.01  -79.93      5.61
      23.47 21.63 -13.6 0     0     0     0     0     0
-0.47 39.85 32.88 0     0     0     0     0     0     0     0     0     0     0
      0     -0.71 -28.86      -2.08 0     71.07 -3.65 13.63 -8.64 2.02  0     0
      0     0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     -28.02      -40.89      58.39 -0.03 -72.54      -3.64
      16.12 7.85  7.15  0     0     0     0     0     0
```

```
/***********************
 * march.sbvh
 ************************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 0.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET -0.0000 -17.5000 -0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -17.0000 -0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
            OFFSET 0.0000 -4.7000 -2.0000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -5.5000 -0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -17.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
            OFFSET 0.0000 -4.7000 -2.0000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 3.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 18.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 6.5000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -11.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -10.5000 -0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
              OFFSET -0.0000 -6.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 18.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -6.5000 -0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -11.0000 -0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -10.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
              OFFSET 0.0000 -6.0000 -0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 18.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 3.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
          OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames:     0
Frame Time: 0.3
```

```
/*********************
 * rotMoveTest.sbvh
 *********************/

HIERARCHY
ROOT Hips
{
  OFFSET 0.0000 0.0000 0.0000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT LeftHip
  {
    OFFSET 3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT RightHip
  {
    OFFSET -3.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightKnee
    {
      OFFSET 0.0000 -17.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightAnkle
      {
        OFFSET 0.0000 -15.5000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        End Site
        {
          OFFSET 0.0000 -3.5000 -1.5000
        }
      }
    }
  }
  JOINT Chest
  {
    OFFSET 0.0000 5.0000 -1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT LeftCollar
    {
      OFFSET 0.0000 13.0000 1.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftShoulder
      {
```

```
      OFFSET 8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT LeftElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT RightCollar
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT RightShoulder
    {
      OFFSET -8.0000 0.0000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT RightElbow
      {
        OFFSET 0.0000 -12.0000 0.0000
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightWrist
        {
          OFFSET 0.0000 -9.5000 0.0000
          CHANNELS 3 Zrotation Xrotation Yrotation
          End Site
          {
            OFFSET 0.0000 -7.0000 0.0000
          }
        }
      }
    }
  }
  JOINT Neck
  {
    OFFSET 0.0000 13.0000 1.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Head
    {
      OFFSET 0.0000 6.5000 0.0000
      CHANNELS 3 Zrotation Xrotation Yrotation
      End Site
      {
        OFFSET 0.0000 7.0000 0.0000
      }
    }
  }
}
```

```
}
MOTION
Frames: 0
Frame Time: 0.2000
```