SPINNER
A concurrent language for programming SPIN

William Beaver[1]
wmb2013@columbia.edu

Marc Eaddy
eaddy@cs.columbia.edu

Gaurav Khandpur
gaurav_k@cs.columbia.edu

Sangho Shin
ss2020@cs.columbia.edu

18 December 2003

[1]team leader

Contents

Chapter 1

Overview

1.1 Background

SPINNER is a general motion control language designed to control SPIN: a performance art installation. Slowly revolving orange wheels are projected onto a hanging grid of translucent Plexiglas circles. The wheels act individually and in groups, joining forces and breaking away in an evolving dance of whole and parts, synchrony and disorder. In SPIN, sounds are produced as if each disc were a phonograph record. As each disc spins fast or slow, backwards or forwards, speeding up or slowing down, its sound follows accordingly. SPIN was created by sound designer and multimedia artist Ben Rubin and presented at the Bumbershoot Visual Arts Exhibition in Seattle, Washington from August 26th to September 1st, 2003.

SPIN is built in the MAX/MSP programming environment; a graphical environment for music, audio, and multimedia. The basic environment that includes MIDI, control, user interface, and timing objects is called Max. Built on top of Max are MSP, a set of audio processing objects that do everything from interactive filter design to hard disk recording, and Jitter, a set of matrix data processing objects optimized for video and 3-D graphics.

To originally control and coordinate the movement of each of SPIN's six discs, a simple, asynchronous, linear scripting language - SPINscript - was created. This language is comprised of a number of initialization parameters and three simple primitive functions: *wait*, *line* and *speed*. The initialization parameters setup audio (gain, assign sound samples to each disc, etc) and define the physics of the system (for example, disc acceleration and deceleration). The primitives are used in concert to control and coordinate the motion of each disc. *Line* tells a particular disc to rotate to a particular angle within a specific timeframe. *Speed* changes the linear acceleration of a disc. By using the *wait* command, a programmer can set discs in motion and then wait a specific time period before issuing the next set of commands; allowing the discs to move in space according to the direction of the line and speed commands. SPINscript is asynchronous. Without the *wait* command, a script attempting to coordinate disc movement by issuing a series of specific motions would in actuality set a disc into one motion which is exactly the additive result of all the commands issued in the script for a disc.

SPINNER is a new language for SPIN. SPINNER is intended to replace SPINscript as the primary interface for expressiveness in SPIN and to solve many of the problems that are inherent in the existing language (see below). SPINNER will generate a file that is compatible with the existing SPINscript language and is subsequently used to control SPIN.

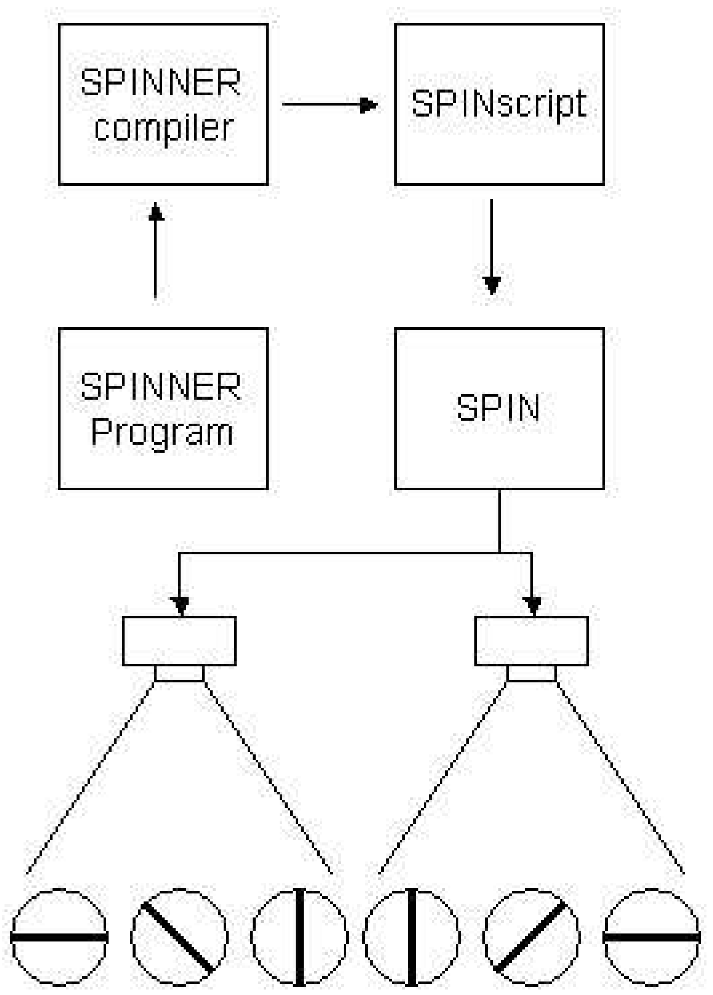

Figure 1.1: SPIN component diagram. SPINNER program read by SPINNER compiler. SPINNER compiler outputs SPINscript which is read by SPIN to control projection of discs.

1.2 The Problem

The original SPINscript language has a number of deficiencies that inevitably limit the expressiveness of individual disc motion and the potential complexity of disc interaction.

1.2.1 Primitive programming model

SPINscript has a strictly sequential control flow. It does not support looping, function calls or jumps. If the user wants to gradually increase the speed of a disc or play the same sequence several times they have to laboriously duplicate the scripting code by hand.

Another problem is that SPINscript does not support variables or constants. This leads to a proliferation of "magic numbers" throughout the script whose intent is impossible to discern. If the artist wants to increase the speed of an entire sequence by a factor of two, she has to calculate and modify each number by hand. This is one area which makes the language very brittle.

1.2.2 Syntax is non-intuitive

SPINscript lines follow the following format:

```
<number>, [disc-]<cmd> <args>;
<number>, --wait-- <ms>;
<number>, --[comment];
```

Here are some examples:

```
1, 6-speed .50 10000;
1, --wait-- 14000;
1, ---------------straight up now;
```

There are several problems with this syntax:

1. The first value *(1,)* is unnecessary. The script language designer has indicated that the <number>, portion isn't used.
2. The disc number that a command operates on is prefixed to the command's name (eg. 6-speed). This syntax is limiting; it does not allow actions on multiple discs, and it is unintuitive.
3. Arguments to functions are not bound to the function name.
4. The wait command does not have the same syntax as other SPINscript commands. It almost looks like a comment.
5. Comments implicitly indicated by the lack of a keyword after "- -" rather than explicitly stated through their own syntax.

1.2.3 Modeling limitations

SPINscript models changing the angles of six independent discs over time. However, this only models part of what the artist is trying to achieve. In many of the "scenes" the artist requires that the behavior of the discs be dependent on each other. For example, a scene might start out with all discs rotating at different speeds. This goes on for several seconds. Then when two discs happen to both be at 0 degrees rotation, the artist wants both discs to synchronize their rotation speed.

Unfortunately, SPINscript does not have a built-in mechanism to perform this synchronization. The artist has to do this calculation manually which is tedious and prone to error. Because of the manual labor involved, scripts must be kept small and simple. This ultimately limits the artist's expressiveness.

1.3 Our Goals

1.3.1 Improved programming model

We will support various methods for control flow; loops, iteration, non-recursive function calls. We will support integer and floating point variables. Additionally, SPINNER provides some built-in functions and operations and the ability to define new functions and procedures using these primitives. For example, support for sine, cosine, and tangent functions could be developed using SPINNER using the basic math operations of addition, subtraction, multiplication, and division. Here is an example:

```
//Start all the discs spinning at different speeds based on sine function
for (double i := 1; getDiscsSize)(); i++) {
   seek(i, sine(i * 1000), i*1000);
}
```

In addition to the for-loop, function call, and variable use, this example also shows how the disc number will become an argument to the function instead of being part of the function name.

1.3.2 Intuitive language syntax

The programmer must be able to transform concepts of the motion of images in space into the language easily. SPINNER supports disc referencing in a consistent manner. Inline comments are handled in a manner more consistent with existing programming languges like C and Java. Use of procedural language constructs provide a natural mechanism for control flow and reference.

1.3.3 Natural high-level semantics

For SPINNER to be successful, it must closely model the problem domain. The programmer must be able to transform concepts of the motion of images in space into the language easily. We will provide a mechanism for synchronizing discs based on different conditions, querying disc state (angle, speed), etc. For the first time the artist will be able to use the language to define behavior dependencies between discs.

For example, let us say that the artist wants to start two discs rotating at different speeds:

```
1, --Start Disc 1 rotating at 2 revolutions per second--;
1, 1-line 2 1000;
1, --Start Disc 2 rotating at 1 revolution per second--;
1, 2-line 1 1000;
```

The artist wants them to rotate like this for 2 seconds:

```
1, --Wait 2 seconds--;
1, --wait-- 2000;
```

Now she wants synchronize the speed of Disc 2 with Disc 1 *when both are at 0 degrees.* How does she do this? Previously, she would need to calculate the exact time this condition would occur. In this example, it would take an additional 1 second before both discs are at 0 degrees. She would then create a wait statement for this amount of time:

```
1, --Wait until both discs are at 0 degrees--;
1, --wait-- 1000;
1, --Now make the speeds the same--;
1, 2-line 2 1000;
```

The synchronization problems centers around the $1, --wait--1000;$ statement. Currently this requires the artist to know the current speeds and starting times of each disc in order to determine their angles at any point during the scene. Our goal is to have our compiler determine this number and allow the artist to specify a conditional that closely models the problem domain.

Here's a pseudocode example of a synchronization statement that the artist could specify in SPINNER:

```
// Wait until both discs are at 0 degrees
waitUntil (getPosition(1) = 0 and getPosition(2) = 0);aitUntil-like
// Now make disc 2 speed equal to disc 1
setSpeed(2, getSpeed(1),0);
```

In order to support a waitUntil command, it is necessary for our compiler to keep track of the speeds of each of the discs *as it is outputting the SPINscript*. We may also have to take into account the time it takes the scripting commands themselves to execute. This is analogous to the amount of clock cycles needed to execute an assembly language instruction.

Part I

Language Reference Manual

Chapter 2

Introduction

This Language Reference Manual (LRM) is organized as follows: this introduction outlines the document organization, briefly discusses the concurrency model required for the language as motivation for the language design, and highlights typographic conventions used. We then discuss the grammar hierarchy for SPINNER, proceed to the lexical structure, discuss types values and variables, names, operators, blocks and statements, expressions, and assignment. Then, we describe the built-in functions to SPINNER and discuss the concurrency model in detail. In appendices, we provide the complete grammars, the beginnings of a function library for the language, and a few example programs.

SPINNER is a language designed for the expression of motion, its interaction with time in a physical system, and the issues of synchronicity that arise in such a system. (see the SPINNER whitepaper for discussion of motivation and target environment.) There are two primary functions needed in the target environment to express motion and time: the ability to seek a disc to a position within a specific duration and the ability to set the speed of a disc independent of setting the position. Upon these two primitives SPINNER adds control over system level settings like gain and volume. SPINNER also provides control over physical system settings that control the analogs of acceleration, deceleration, and motion/sound interaction. Features such as basic control flow (looping, branching, etc.), variables, access to primitive data structures, and access to disc level state information like current speed and position gives the SPINNER programmer levels of control that were previously unattainable. Additionally, SPINNER supports a concurrent processing model which allows process branching and interprocess communication that greatly eases the programming of multiple objects in tandem. (See Chapter 12 for more details.)

Throughout this document, references to one disc can be applied to `all` discs unless specified. The `typewriter font` is used for all keywords.

Chapter 3

Language construction

SPINNER is defined using Context Free Grammars (CFGs). These CFGs define the lexical conventions of the language, which are used to take the input (a series of ASCII characters), remove white space, and translate it into a sequence of tokens. The CFGs that define the syntactic grammar of SPINNER use the tokens defined by the lexical grammar to describe how the sequence of tokens form syntactically correct programs in the language.

Chapter 4

Lexical conventions

4.1 Line Terminator

Lines are terminated by the ASCII characters CR, or LF, or CR LF. The two characters CR immediately followed by LF are counted as one line terminator, not two.

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

4.2 Tokens

Tokens in SPINNER define comments, identifiers, keywords, constants, literals, line terminators, operators, and separators in the language. White space is ignored in the system (as defined below) except where it is used to separate tokens.

4.3 Comments

SPINNER supports two types of comments. /* introduces a comment, which terminates with the characters */. // is also used to introduce a comment which terminates with the line separator (see below). Comments do not nest. /* and */ have no special meaning in comments that begin with //. // has no special meaning in comments that begin with /*.

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

4.4 White space

White space is defined as the ASCII space and horizontal tab characters, as well as line terminators.

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

4.5 Identifiers

An identifier is a sequence of letters and digits. First character must be a letter, and the underscore character counts as a letter. Case matters. Identifiers may have any length. Identifiers may not be a reserved keyword, a `boolean` literal, or the `null` literal.

```
protected LETTER : ('a'..'z' | 'A'..'Z') ;
: LETTER (LETTER | DIGIT | '_')* ;
```

4.6 Keywords

The following are reserved as keywords in the system and may not be used otherwise:

`all`	`double`	`repeat`
`boolean`	`else`	`return`
`break`	`for`	`then`
`while`	`func`	`void`
`do`	`if`	`not`
`and`	`or`	`until`

4.7 Literals

4.7.1 `double` literal

`double` literals consist of an integer part, a decimal part, and a fraction part. Integer and fraction parts each consist of a sequence of digits. The decimal part and fraction part are together optional.

```
protected DIGIT : '0'..'9' ;
ID options { testLiterals = true; }
: LETTER (LETTER | DIGIT | '_')* ;
NUMBER : (DIGIT)+ ('.' (DIGIT)+)?
       | '.' (DIGIT)+
;
```

4.7.2 `boolean` literal

The `boolean` type has two values, represented by the literals `true` and `false`, formed from ASCII letters.

```
boolean:
      ["true" | "false"]
```

4.7.3 `null` literal

The `null` type has one value, the `null` reference, represented by the literal `null`, which is formed from ASCII characters. A `null` literal is always of the `null` type.

```
null:
     "null"
```

4.7.4 String literal

A string literal is a sequence of characters surrounded by double quotes. String literals do not contain newline or double quotes.

```
STRING : '"'! ('"' '"'! | ~('"'))* '"'! ;
```

4.8 Separators

The following nine ASCII characters are the separators (punctuators):

```
(   )   {   }   [   ]   ;   ,   .
```

4.9 Operators

The following seventeen objects represent operators:

```
<  >  <=  >=  =  not
+  -  *  /  %
++  --  and  or
:=  ||
```

Chapter 5

Types, values and variables

5.1 Introduction

SPINNER is a strongly typed language; which means that every variable and every expression has a type that is known at compile time. Types limit the values that a variable can hold or that an expression can produce, limit the operations supported on those values, and determine the meaning of the operations. Strong typing helps detect errors at compile time.

5.2 Data types

SPINNER supports the `double`, `boolean`, and array data types.

5.2.1 `double`

`double` variables consist of an integer part, a decimal part, and a fraction part. Integer and fraction parts each consist of a sequence of digits. The decimal part and fraction part are together optional. A `double` can be optionally signed.

```
protected DIGIT : '0'..'9' ;
ID options { testLiterals = true; }
: LETTER (LETTER | DIGIT | '_')* ;
NUMBER : (DIGIT)+ ('.' (DIGIT)+)?
       | '.' (DIGIT)+
;
```

5.2.2 `boolean`

The `boolean` type has two values, represented by the literals `true` and `false`, formed from ASCII letters.
`boolean` → [TRUE | FALSE]

5.2.3 Array

A SPINNER array can contain any of the data types supported, including other arrays. Arrays are indexed $1 \ldots N$ where N is the size of the array. Declaration of an array type later allows programmer to access the getSize() function which returns the number of elements in the referenced array. SPINNER array elements are referenced by array name[index] where the index refers to an element of the array. If index does not refer to a valid element in the array (i.e. the index is not within $1 \ldots N$) a compiler error occurs.

5.3 Variables

A variable is a storage location and has an associated type. A variable always contains a value that is an assignment compatible with its type.

5.3.1 Naming

Variable definitions consist of a type declaration and variable name and an optional initialization. SPINNER does not support multiple variable declarations of the same type in the same declaration; each variable must be individually declared. SPINNER does not support duplicate variable name declarations within the same scope and will throw an error at compile time.

Example:

```
type varName;

type varName := value;
```

5.3.2 Scope

Global

SPINNERs global scope is available to all procedures, functions and scopes and is defined as everything within the topmost structure of the program.

Local

SPINNER provides local scoping. Local scope is maintained for each procedure, function and block statement and propagates to their children process/scopes. Local scope is the present scope plus child scopes.

5.3.3 Initial values

`boolean` values are initialized to `false` if no value is provided at declaration.
`double`s are initialized to 0.0 at instantiation if no value is provided.
Array elements are initialized to `null` at declaration.

Chapter 6

Conversion and promotion

6.1 `boolean` → `double`

`boolean` values are converted as follows: `true` → 1 and `false` → 0 when used in places where type `double` is expected.

6.2 `double` → `boolean`

`double` values are converted to `boolean` values as follows: $-\infty \ldots 0 \rightarrow$ `false` and $1 \ldots +\infty \rightarrow$ `true`.

6.3 `double` truncation

`double` are used in situations where traditional integer values are required (as array indices, disc number references, etc.). When this occurs, SPINNER truncates the value by dropping the decimal and fraction part of the number if it was provided. If only the decimal and fractional part are provided, SPINNER interprets this value as 0 (zero).

6.4 `null`

`null` value is not converted and will throw error at compile time if used when expecting a declarable type.

Chapter 7

Names

Names are used to refer to entities declared in a program. A declared entity is a parameter, a local variable, or a function. Names in programs are simple, consisting of a single identifier. Every declaration that introduces a name has a scope, which is the part of the program text within which the declared entity can be referred to by a simple name. The name of a field, parameter, or local variable may be used as an expression. The name of a function may appear in an expression.

7.1 Declarations

A declaration introduces an entity into a program and includes an identifier that can be used in a name to refer to this entity. A declared entity is one of the following:

- A function
- A parameter
- A local variable, one of the following:
  - A local variable declared in a block
  - A local variable declared in a `for` statement

7.2 Scope of a declaration

The scope of a declaration is the region of the program within which the entity declared by the declaration can be referred to using a simple name (provided it is visible). A declaration is said to be in scope at a particular point in a program if and only if the declaration's scope includes that point.

The scoping rules for various constructs are given here:

1. The scope of a type imported by a import declaration is the same as if the declarations occurred within the program itself. For example, a variable which was global to the program being imported is global in the program it was imported into as import statements can only occur within the global scope.

2. The scope of a parameter of a function is the entire body of the function.

3. The scope of a local variable declaration in a block is the rest of the block in which the declaration appears, starting with its own initializer and including any further declarators to the right in the local variable declaration statement.

4. The scope of a local variable declared in the initialization part of a `for` statement includes all of the following:

   - Its own initializer
   - Any further declarators to the right in the initialization part of the `for` statement
   - The Expression and update parts of the `for` statement
   - The contained Statement

7.3 Naming conventions

7.3.1 Function names

Function names should be verbs or verb phrases, in mixed case, with the first letter lowercase and the first letter of any subsequent words capitalized. Here are some additional specific conventions for function names:

- Functions to get and set an attribute that might be thought of as a variable V should be named getV and setV.

- A function that returns the length of something should be named size, as in arrays.

- A function that tests a boolean condition V about an object should be named isV.

7.3.2 Constant names

Note: SPINNER does not support constants and as such will not protect values of variables as you would constants. By naming variable according to a standard they can be easily recognized by programmers and appropriate care taken to protect the integrity of their values.

The names of variables to be treated as constants should be a sequence of one or more words, acronyms, or abbreviations, all uppercase, with components separated by underscore "_" characters. Constant names should be descriptive and not unnecessarily abbreviated. Conventionally they may be any appropriate part of speech. Examples of names for constants include MIN_VALUE, MAX_VALUE, MIN_RADIX, and MAX_RADIX.

7.3.3 Local variable and parameter names

Local variable and parameter names should be short, yet meaningful. They are often short sequences of lowercase letters that are not words. Of course, variable and parameter names cannot be keywords.

Chapter 8

Blocks and statements

8.1 Blocks

A block is a sequence of statements, local class declarations and local variable declaration statements within braces.

```
main_statement
        :outer_statement
        |func_definition
;

outer_statement
        :statement
        |declaration SEMI
        |parallel_statement
;

statement
        :if_statement
        |for_statement
        |while_statement
        |repeat_statement
        |assignment SEMI!
        |func_call SEMI!
        |create_disk SEMI!
        |SEMI!
;

inner_statement
        :statement
        |break_statement
        |LBRACE! (main_statement)* RBRACE!
```

```
;
```

A block is executed by executing each of the local variable declaration statements and other statements in order from first to last (left to right). If all of these block statements complete normally, then the block completes normally. If any of these block statements complete abruptly for any reason, then the block completes abruptly for the same reason.

8.2 Local variable declaration statements

```
variable_declaration
        : type (ID | assignment_with_decl | array_decl)
;

type
        : "double"
        | "boolean"
        | "void"
;
```

Every local variable declaration statement is immediately contained by a block. Local variable declaration statements may be intermixed freely with other kinds of statements in the block.

A local variable declaration can also appear in the header of a `for` statement. In this case it is executed in the same manner as if it were part of a local variable declaration statement.

8.2.1 Local variable declarators and types

Each declarator in a local variable declaration declares one local variable, whose name is the Identifier that appears in the declarator. Each declarator in a local variable declaration declares one local variable, whose name is the Identifier that appears in the declarator.

Examples:

```
type varName;  \\variable without initialization

type varName = value;  \\variable with initialization

type varName[size];  \\array declaration
```

8.2.2 Scope of local variable declarations

The scope of a local variable declaration in a block is the rest of the block in which the declaration appears, starting with its own initializer and including any further declarators to the right in the local variable declaration statement. The name of a local variable v may not be redeclared as a local variable of the directly enclosing function, block or initializer block within the scope of v, or a compile-time error occurs.

The scope of a local variable declared in a `for` statement is the rest of the `for` statement, including its own initializer.

8.2.3 Execution of local variable declarations

A local variable declaration statement is an executable statement. Every time it is executed, the declarators are processed in order from left to right. If a declarator has an initialization expression, the expression is evaluated and its value is assigned to the variable. If a declarator does not have an initialization expression, then a SPINNER initializes the variables as defined in section 5.3.3.

Each initialization (except the first) is executed only if the evaluation of the preceding initialization expression completes normally. Execution of the local variable declaration completes normally only if evaluation of the last initialization expression completes normally; if the local variable declaration contains no initialization expressions, then executing it always completes normally.

8.3 Statements

There are many kinds of statements in the SPINNER programming language. Most correspond to statements in the Java, C, and C++ languages. As in C and C++, the `if` statement of the Java programming language suffers from the so-called "dangling else problem". SPINNER, like the Java programming language, C, and C++ and many programming languages before them, arbitrarily decree that an `else` clause belongs to the innermost `if` to which it might possibly belong. This rule is captured by the following grammar:

```
statement
        :if_statement
        |for_statement
        |while_statement
        |repeat_statement
        |assignment SEMI!
        |func_call SEMI!
        |create_disk SEMI!
        |SEMI!
;

inner_statement
        :statement
        |break_statement
        |LBRACE! (main_statement)* RBRACE!
;
```

8.3.1 Empty statement

An empty statement does nothing.

```
EmptyStatement:
        ;
```

Execution of an empty statement always completes normally.

8.3.2 Expression statements

Certain kinds of expressions may be used as statements by following them with semicolons. An expression statement is executed by evaluating the expression; if the expression has a value, the value is discarded. Execution of the expression statement completes normally if and only if evaluation of the expression completes normally. SPINNER allows only certain forms of expressions to be used as expression statements. (see Chapter 9)

8.3.3 `if then` | `if then else`

In both forms of the `if` statement, the expression is evaluated, and if it evaluates equal to `true`, the first substatement is executed. In the second form, the second substatement is executed if the expression is `false`.

```
if_statement
        : "if" LPAREN! expr RPAREN! "then"! inner_statement
        (options {greedy=true;} : "else"! inner_statement)?
;
```

8.4 Iteration

8.4.1 `while`...`do`

The substatement is executed repeatedly so long as the value of the expression remains equal to `true`. The test precedes one iteration of the statement.

```
while_statement
        : "while" LPAREN! expr RPAREN! "do" inner_statement
;
```

8.4.2 `repeat`...until

The substatement is executed repeatedly so long as the value of the expression remains equal `true`. The test follows after at least one iteration of the statement.

```
repeat_statement
        : "repeat" inner_statement "until" LPAREN! expr RPAREN!
;
```

8.4.3 `for`

The first statement is evaluated once, establishing the initialization of the loop. There is no restriction on its type. The second is an expression; it is evaluated before each iteration and if it becomes `false`, the `for` statement is terminated. The third expression is evaluated after each iteration and thus defined the reinitialization of the loop.

```
for_statement
        : "for" LPAREN! (for_init)? SEMI! (expr)? SEMI! (for_increment)? RPAREN! inner_statement
;
```

```
for_init
        :assignment
        |declaration
;

for_increment
        :assignment
;
```

8.4.4 `break` statement

`break` statement may appear in any blockstatement (except the global statement) and it terminates the execution of the smallest enclosing statement and returns control to the statement following the terminated statement.

```
break_statement
        : "break" SEMI!
;
```

8.4.5 `return` statement

A `return` statement returns control to the invoker of a function.

```
return_stmt
        : "return" (expr)?
;
```

A `return` statement with no Expression must be contained in the body of a function that is declared, using the keyword `void`, not to return any value. A `return` statement with no Expression transfers control to the invoker of the function that contains it. A `return` statement with no Expression always completes abruptly, the reason being a `return` with no value.

A `return` statement with an Expression must be contained in a function declaration that is declared to return a value or a compile-time error occurs. The Expression must denote a variable or value of some type T, or a compile-time error occurs. The type T must be assignable to the declared result type of the function, or a compile-time error occurs.

A `return` statement with an Expression transfers control to the invoker of the function that contains it; the value of the Expression becomes the value of the function invocation. More precisely, execution of such a `return` statement first evaluates the Expression. If the evaluation of the Expression completes abruptly for some reason, then the `return` statement completes abruptly for that reason. If evaluation of the Expression completes normally, producing a value V, then the `return` statement completes abruptly, the reason being a `return` with value V.

It can be seen, then, that a `return` statement always completes abruptly.

8.5 Concurrency operator ||

Concurrent execution paths are created by setting block statements off and joining them with ||. When a block containing thread is encountered, SPINNER forks and concurrently processes each block. The entire block completes when all threads terminate. || binds tightly to adjoining block statements and block statements *only*. For example

```
{blockstatement} || {<blockstatement}
```

is a valid concurrency operator.

```
{blockstatement} || setSpeed(i, 4, 4000);
```

is an error (parallel block with standalone function) as is the following:

```
 function; || setSpeed(i, 4, 4000);
```

(a standalone function parallel to another standalone function)
This is valid:

```
{function} || {setSpeed(i, 4, 4000); }
```

because each function is enclosed as a blockstatment.

Termination of a series of concurrent statements occurs when all statements have completed execution. Until all the processes complete or are terminated, program flow will halt. Where possible, the compiler will indicate deadlock situations, but care should be taken by the programmer to avoid such situations.

Concurrency can be nested as long as proper block statement nesting syntax is used. When nested, control lives in the bottommost processes of the process fork tree and flows upwards.

For a full description of SPINNER's concurrency model, see Chapter 12.

Chapter 9

Expressions

When an expression in a program is evaluated (executed), the result denotes one of three things:

1. A variable (in C, this would be called an lvalue)
2. A value
3. Nothing (the expression is said to be `void`)

Evaluation of an expression can also produce side effects, because expressions may contain embedded assignments, increment operators, decrement operators, and method invocations.

An expression denotes nothing if and only if it is a function call that invokes a function that does not return a value, that is, a function declared `void`. Such an expression can be used only as an expression statement, because every other context in which an expression can appear requires the expression to denote something. An expression statement that is a functional call may also invoke a function that produces a result; in this case the value returned by the function is quietly discarded.

Value set conversion is applied to the result of every expression that produces a value.

9.1 Evaluation order

In SPINNER, operands of operators appear to be evaluated in a specific evaluation order, namely, from left to right.

The left-hand operand of a binary operator appears to be fully evaluated before any part of the right-hand operand is evaluated.

Every operand of an operator (except the conditional operators `and or`) appears to be fully evaluated before any part of the operation itself is performed.

Evaluation respects parentheses and precedence.

Argument lists are evaluated left-to-right

9.2 Precedence

SPINNER has the following precedence levels (from highest to lowest):

```
or and
*  /  mod(%)
+  -  ++  --
< > <= >= =
:=
```

9.3 Array creation

An array instance creation expression is used to create new arrays.

```
declaration:
            type ID[expr];
```

An array creation expression creates an object that is a new array whose elements are of the type specified by the type.

The type of each dimension expression within a expr must be an integral type, or a compile-time error occurs. Each expression undergoes promotion which is double truncation of the decimal part, or a compile-time error occurs. SPINNER does not support array intialization at declaration. If an array initializer is provided, an error is generated at compile time.

9.3.1 Array access expressions

An array access expression refers to a variable that is a component of an array.

```
access:
       ID[expr]
```

An array access expression contains two parts, the array reference ID (before the left bracket) and the index expression (within the brackets). Note that the array reference ID may be a name or any primary expression that is not an array creation expression. The type of the array reference expression must be an array type (call it T[], an array whose components are of type T) or a compile-time error results. Then the type of the array access expression is T.

The index expression undergoes unary numeric promotion. If the index type is `double`, the decimal part is truncated (unofficially promoted to a integer.)

The result of an array reference is a variable of type T, namely the variable within the array selected by the value of the index expression.

An array access expression is evaluated using the following procedure:

- The array reference expression is evaluated. If this evaluation completes abruptly, then the array access completes abruptly for the same reason and the index expression is not evaluated.
- Otherwise, the index expression is evaluated. If this evaluation completes abruptly, then the array access completes abruptly for the same reason.
- Otherwise, if the value of the array reference expression is `null`, then a NullPointer error is raised.
- Otherwise, the value of the array reference expression indeed refers to an array. If the value of the index expression is less than zero, or greater than or equal to the array's length, then an ArrayIndex-OutOfBounds error is raised.

- Otherwise, the result of the array access is the variable of type T, within the array, selected by the value of the index expression.

9.3.2 Array assignment expression

Array assignment expressions come in two flavors. One uses explicit indexing, the other implicit indexing. To assign values to an array explcitly, each element of the array is indexed individually and a value assigned. For example:

```
array[1]:= 3.0; //set the first element of array to 3.0

array[i]:= k; //set the ith element of array to value stored in variable k
```

If the array index is out of bounds, a compile error will occur. If the assignment types are not consistent, SPINNER attempts implcit type conversion (Chapter 6). If this fails, a compile time error will occur.

For implicit array assignment, all or some of the array elements are referenced and assigned. For example:

```
array[]:= {3.0, 4.0}; //set the first and second elements of array to 3.0 and 4.0
```

The assignment begins from the lowest (leftmost) index and places the values in the bracketed statement accordingly. If the number of elements be assigned exceeds the number of elements in the array, a compile error occurs.

9.4 Function invocation expressions

A function invocation expression is used to invoke a function.

```
func_call
        : ID LPAREN (params)? RPAREN
;
```

9.5 Postfix expressions

9.5.1 Postfix increment (x++)

A unary expression proceeded by a ++ operator is a postfix increment expression. The result of the unary expression must be a variable of a numeric type. Postfix increment in SPINNER means the value of the variable being incremented is incremented after the variable is used.

9.5.2 Postfix decrement (x−−)

A unary expression proceeded by a −− operator is a postfix decrement expression. The result of the unary expression must be a variable of a numeric type. Postfix decrement in SPINNER means the value of the variable being decremented is decremented after the variable is used.

9.6 Unary operators

SPINNER only support one unary operator: logical negation

9.6.1 Logical negation operator `not`

An expression prefixed with the logical negation operator will be evaluated and then the result negated its result.

9.7 Multiplicative operators

9.7.1 Multiplication operator *

The binary * operator performs multiplication, producing the product of its operands. Multiplication is a commutative operation if the operand expressions have no side effects.

9.7.2 Divide operator /

The binary / operator performs division, producing the quotient of its operands. The left-hand operand is the dividend and the right-hand operand is the divisor.

9.7.3 Remainder operator %

The binary % operator is said to yield the remainder of its operands from an implied division; the left-hand operand is the dividend and the right-hand operand is the divisor. In C and C++, the remainder operator accepts only integral operands, but in the SPINNER programming language, it also accepts floating-point operands.

9.8 Additive operators

9.8.1 Addition operator +

The binary + operator performs addition when applied to two operands of numeric type, producing the sum of the operands. Addition is a commutative. Addition is associative.

9.8.2 Subtraction operator -

The binary - operator performs subtraction, producing the difference of two numeric operands.

9.9 Relational operators

SPINNER supports the following relational operators:

```
> <  <= >= =
```

The type of a relational expression is always `boolean`:

- The value produced by the $<$ operator is `true` if the value of the left-hand operand is less than the value of the right-hand operand, and otherwise is `false`.
- The value produced by the $\leq$ operator is `true` if the value of the left-hand operand is less than or equal to the value of the right-hand operand, and otherwise is `false`.
- The value produced by the $>$ operator is `true` if the value of the left-hand operand is greater than the value of the right-hand operand, and otherwise is `false`.
- The value produced by the $\geq$ operator is `true` if the value of the left-hand operand is greater than or equal to the value of the right-hand operand, and otherwise is `false`.
- The value produced by the $=$ operator is `true` if the value of the left-hand operand is equal to the value of the right-hand operand, and otherwise is `false`.

9.10 Logical operators

9.10.1 Logical `and`

For `and`, the result value is `true` if both operand values are `true`; otherwise, the result is `false`. The `and` operator is lazy; meaning that if the first operand is `false`, the statement will immediately evaluate to `false` without evaluation of the second operand. It is syntactically left-associative (it groups left-to-right).

9.10.2 Logical `or`

The result value is `true` if either of the operand values are `true`. The `or` operator is lazy; meaning that if the first operand is `true`, the statement will immediately evaluate to `true` without evaluation of the second operand. It is syntactically left-associative (it groups left-to-right).

9.11 Assignment operator :=

Assignment operator := places the value of the right operand into the left operand. Type conversion (as explained in Chapter 6) attempts to fix assignments. If the assignment fails, a compile time error will occur.

Chapter 10

Functions

10.1 Declaration and format

A function in SPINNER is named block of statements that accepts an optional parameter list of input and returns an optional value using the `return` statement (section 8.4.5).

Functions are declared by first identifying its return type. If a function is not going to return anything, it should declare type `void`. Then the keyword `func` is used to identify that this is a function declaration followed by the name of the function. Finally, a list of parameters are declared for the function. These parameters become variables accessible within the local scope of the function. After the function declaration, a block statement contains the statements and expressions of the function.

```
func_definition
        : type "func" ID LPAREN! (args)? RPAREN! LBRACE! func_body RBRACE!
;

func_body
        : (main_statement)*
;

args
        : arg (COMMA! arg)*
;

arg
        : type ID
;

params
        : expr (COMMA! expr)*
;
```

Every function must terminate abruptly through the use of the `return` statement. If a function declares a return type of `void`, then a `return` statement with no Expression must be called to terminate the function.

If a function declares a return type of some SPINNER data type, then the `return` statement must be called with an Expression than matches the declared return type. If the Expression returns a type that is inconsistent with the declared return type, SPINNER will attempt to promote the return value using its promotion and conversion rules (Chapter 6). If promotion or conversion is not possible, the function will throw an error.

Chapter 11

Built-in SPINNER functions

SPINNER contains a number of function which are native to the language. These are provided here as a reference. Each function can be called as a standalone expression statement or used in conjunction with other statements. For each, we define the return type, name, parameter list, process description and some examples.

11.1 seek

command	seek
return type	`void`
syntax	seek([discnum | `all`], SeekPosition, Time);
parameters:	

[*discnum* | `all`] is disc number (truncated `double`) that this command applies to. `all` is a valid disc reference for all the discs.

SeekPosition is the absolute numerical position (`double`) that the disc should seek to. 0 is home position, 1.0 is one revolution from home, 2.0 is two revolutions from home, etc. Negative values for SeekPosition indicate counter clockwise rotation.

Time is the number of milliseconds (truncated `double`) the disc has to get to SeekPosition.

The seek command specifies the position a disc should seek from it's present position and the timeframe in which it has to do it.

The seek command is synchronous. It effectively sets a disc's velocity and then waits for TimeN milliseconds. After completion of line command, the disc will stop.

Example

```
//disc 1, spin one clockwise rotation to position 1 in 1000ms=1s and then stop
seek(1,1,1000);
```

Seek commands issued with a wait break between them will behave as follows: the first issuance of the command establishes the position to seek to and the time in which to do it. A wait(N) command allows this motion to proceed for N ms, then the next line command is issued and the disc immediately executes this commands. Disc position and speed when command two is issued are relative to the state after executing command one for N ms. Seek (and setSpeed) commands issued sequentially without intervening wait statements result in the execution of the last command.

11.2 setSpeed

command	setSpeed
return type	`void`
syntax	setSpeed([DiscNum | `all`], TargetSpeed, RampTime);

parameters:

DiscNum is the disc number that this command applies to. `all` is a valid disc reference.
TargetSpeed is the target speed in revolutions per second (`double`).
RampTime is the number of milliseconds (`double`) you have to achieve the TargetSpeed.

setSpeed is an acceleration function. It tells a disc to spin at TargetSpeed revolutions per second and to get to the TargetSpeed in RampTime seconds. Acceleration to TargetSpeed is linear. setSpeed commands are absolute; execution of the command replaces the motion established by a previous setSpeed (or seek)command.

Examples:

```
//disc 1 spin at .75 revs/s and take 10 secs to get there.
setSpeed(1, .75, 10000);

//all discs spin at -.5 revs/ sec [.5rev/sec ccw] and take 5 secs to get there
setSpeed(all, -.5,  5000);

//all discs spin at 2 rev/sec and take 4.5 secs to get there
setSpeed(all, 2.0, 45000);
```

11.3 getSpeed

getSpeed(discnum) returns the `double` value of the reference disc's speed at the time the function is called. If the disc is spinning counter clock-wise, the speed returned will be negative.

11.4 getPosition

getPosition(discnum) returns the `double` value of the disc's position at the time the function is called. Position is reported as number of revolutions away from the zero (0) position. Counter clock-wise revolutions report a negative value.

11.5 wait

Wait(time) takes an integer constant or variable and returns `void`. It instructs the system to pause the current branch for the time indicated (in milliseconds). The time value is truncated if `double` is provided.

11.6 waitUntil

waitUntil(condition, timeout)

The waitUntil function takes a `boolean` expression argument and a timeout value. Its purpose is to block the currently executing branch until the `boolean` expression evaluates to `true` or the timeout expires. This is useful for waiting until a certain condition occurs, for example, waiting for the angle of disc 1 to be 30 degrees, or for synchronizing the actions of two or more discs.

You can specify a timeout value: $= \; -1$ (Infinity) $\geq \; 0$ (Number of milliseconds to wait before timing out)

1. When the boolean expression evaluates to `true`, waitUntil stops blocking and returns the number of milliseconds spent blocked as an integer. The value returned will never be greater than the timeout value unless timeout is -1 (infinity).
2. The simulator will attempt to evaluate the waitUntil condition.
   - If `true`, the instruction ptr for that branch is incremented.
   - Otherwise, it is not incremented. This means that we will continue evaluating the waitUntil conditional until it becomes `true`, possibly forever.
3. While one branch is "stuck" until the condition is `true`, other branches continue executing as normal. When waitUntil evaluates to `true`, the simulator will output a SPINscript wait statement for the time it took the condition to be evaluated to `true`. (Actually, the wait statement may be interleaved with other branch output.)

example:

```
{
   seek(1, 10, 10000);
   waitUntil(getAngle(2) = 180, -1);
   setSpeed(1, getSpeed(2));
}
||
{
   seek(1, 10, 5000);
   wait(10000);
}
```

11.7 setGain

command	setGain
return type	`void`
syntax	setGain([DiscNum | `all` | "master"], val) ;
parameters:	

[*DiscNum* | *`all`* | *"master"*] is the disc number that this command applies to. `all` and "master" are a valid disc references.

val is a continuous value (`double`) on (0,1) to set gain.

setGain sets the gain for the appropriate object (disc or master). setGain returns no value.
Examples

```
setGain(1, 1.);  //set disc 1 gain to 1.0

setGain(all, 0.25); //set all disc's gain to .25

setGain(master, 0.22); //set system gain to .22
```

11.8 getGain

syntax getGain(discnum | master]); getGain returns a `double` indicating the gain value for the requested object or `null` if the gain value is unknown.

11.9 setFile

setFile([discnum | `all`], "filename"); setFile indicates the sound file for the appropriate object (disc or master). setFile returns no value.

11.10 isFileSet

isFileSet(discnum); isFileSet returns a `boolean` indicating whether a file has been set for a particular object. `true` indicates that a file has been set. `false` indicates a file has not been set. `all` is NOT a valid parameter for discnum.

11.11 setReverb

command	setReverb
return type	`void`
syntax	setReverb([“send” | “return”], val);

parameters:
[*“send”* | *“return”*] is string literal used to set appropriate sub feature of reverb. required.
val is continuous value (`double`) to set reverb to. required.

setReverb sets the system reverb. setReverb returns no value.
Examples

```
setReverb(``send'', 1.); //set send reverb to 1.0

setReverb(``return'', 0.25); //set return reverb to .25)
```

11.12 getReverb

getReverb([send | return]);

Get reverb returns the reverb value of the appropriate sub feature of reverb. If the sub feature was not set, getReverb returns `null`. [*send* | *return*] are literals that refer to the specific sub feature of getReverb.

11.13 setSpinMult

command	setSpinMult
return type	`void`
syntax	setSpinMult([discnum | `all`], val);

parameters:

[*discnum* | ***`all`***] is the disc number that this command applies to. `all` is a valid disc references.

val is a value (`double`) to set the spinMult to.

The sound and the rotation in SPIN are locked together, and spinmult is like the gear ratio between them. When spinmult is 1.0, then the sound sample will play back at its original speed when the wheel is turning at 1.0 RPS. When the wheel turns at 2.0 RPS, the sample will play back at double speed and at a higher pitch (1 octave higher, which is double pitch). At .5 RPS, the sound would play back at half speed/half pitch. at -1.0 RPS, the sound will play back at normal speed, but backwards. When spinmult is .5, then the sound sample will play back at 1/2 speed and 1/2 pitch when the wheel is turning at 1.0 RPS, etc.

11.14 getSpinMult

getSpinMult(discnum);

for a referenced object, getSpinMult returns the `double` value of the spinMult. If spinMult value is unknown, getSpinMult returns `null`.

11.15 setSampleStart

command	setSampleStart
return type	`void`
syntax	setSampleStart([discnum | `all`], val);

parameters:

[*discnum* | ***`all`***] is disc number that this command applies to. `all` is a valid disc reference.

val is the point in time within the defined sound file that you would like the sample to start.

see setLength (section 11.17) for description.

11.16 getSampleStart

getSampleStart(discnum);

Returns the sample start `double` value for the disc referenced. If the value is not known, returns `null`.

11.17 setLength

command	setLength
return type	`void`
syntax	setLength([discnum | `all`], val);

parameters:

[*discnum* | *`all`*] is disc number that this command applies to. `all` is a valid disc reference.
val is the length (in ms) of the sample in sound file established by setFile (section 11.9).

setSampStart and setLength define the part of the sample within the file that will actually be used. Both values are milliseconds.

Example:

If you have a sound file that is 10 seconds long, but only want to use a 1-second portion of that file that starts 2.5 seconds from the start of the file. So setSampStart with val = 2500 and setLength val = 1000; This will give a sample 1 second long starting at 2.5 seconds in the file.

11.18 getLength

getLength(discnum);

Returns the `double` value of the length assigned by setLength. If no value has been set, returns `null`.

11.19 setSlide

command	setSlide
return type	`void`
syntax	setSlide([discnum | `all`], ["up" | "down"], val);

parameters:

[*discnum* | *`all`*] is the disc number that this command applies to. `all` is a valid disc reference.
[*"up"* | *"down"*] is the string literal which determines if you are addressing acceleration (up) or deceleration (down).
val is the value (`double`) you want to set the slide value to.

slide up and slide down modify the behavior of "seek". Normal seek behavior described in this doc is `true` when both of these values are set to 0.

Higher values effectively smooth out the acceleration and deceleration.

slide up only affects positive (or upward) movements, and slide down only negative/downward.

These setting ONLY affect movement and acceleration/decelerations that are caused by seek commands – slide up/slide down have no affect on speed or acceleration/deceleration caused by setSpeed commands.

With these settings = 0, the result of a seek command is that the disc jumps from a standstill to the speed determined by the position/time values, and when the position is reached, the disc lurches to an instantaneous stop, and the arrival time is precisely what was specified in the seek command.

With these settings > 0, the disc will ramp up gradually to its cruising speed as it seeks towards its destination position, then decelerate gradually as it approaches.

11.20 getSlide

getSlide(discnum [up | down]);

Return the slide value set for the appropriate disc reference. If no value is known, return `null`.

11.21 setSub

command	setSub
return type	`void`
syntax	setSub([send | return], val);

parameters:

[*send* | *return*] are literals used to set appropriate sub feature of Sub. required.

val determines how much of the sound is sent to the subwoofer – 0.0 is none, 1.0 is all – and these are effectively the bounds of this param.

setSub determines how much of the sound for the system is sent to the subwoofer.

11.22 getSub

command	getSub
return type	`double`
syntax	getSub([send | return]);

parameters:

[*send* | *return*] is literal used to set appropriate sub feature of reverb. required.

Returns value set by setSub. `null` if not set in this SPINNER program.

11.23 createDiscs

createDiscs[val];

createDiscs takes a `double` val and truncates fraction part if present. This values indicates the number of discs available in the system. createDiscs returns no value.

11.24 getDiscsSize

getSize();

getDiscsSize returns the number of discs initialized by the system. If the discs have not been initialized, getDiscsSize returns `null`.

11.25 getSize

getSize(arrayreference);

getSize returns the number of elements in the referenced array. If the the reference has not been initialized, getSize returns `null`.

11.26 getTime

getTime returns the absolute time since beginning of simulation as tracked by SPINNER. Time returned in miliseconds.

11.27 Import

import(“filename”);

Imports a SPINNER file inline and processes. If filename not found, runtime error is thrown. Import statements MUST occur in the first line of any SPINNER program else a compile error will occur.

Chapter 12

Concurrent processing

Most of the discussion in the previous sections dealt with rules for processing a single statement or expression at a time. SPINNER supports the concept of concurrent processing where multiple, concurrent processing paths can execute simultaneously. This section discusses this model.

12.1 SPINscript - Linear execution model

In SPINscript, SPINNER's target language, statements are executed one after another until the end of the file. SPINscript does not support looping, jumping, function calls or parallel execution. This limits the expressive power of the language, and therefore the programmer/artist.

Here is an example of something that is easy to do in SPINscript:

> Start discs 1 and 2 spinning at different speeds
>
> Wait for 30 seconds
>
> Change speeds of discs 1 and 2

Here is an example of something that is difficult to do in SPINscript:

> Start disc 1 spinning with a speed defined by a sine wave
>
> Start disc 2 spinning with a speed defined by a cosine wave
>
> Wait for 10 seconds
>
> Wait until the speed of disc 1 and disc 2 are nearly equal
>
> Change speeds of disc 1 and 2 to be defined by acceleration function

SPINscript has an execution model which only "enables" two types of speed functions, a linear function via Line and an acceleration function via Speed. It is possible to implement more complicated speed functions in SPINscript, just as it is possible to write object-oriented constructs in C. It is just not easy. If the artist wanted to implement the sine and cosine speed functions in SPINscript they would need to manually "interleave" Speed and Wait commands:

```
Speed-1, 8.41, 1000     ; sine of 1 * 10 rev/s for 1 sec
Speed-2, 5.40, 1000     ; cosine of 1 * 10 rev/s for 1 sec
Wait 1000;
Speed-1, 9.09, 1000     ; sine of 2 * 10 rev/s for 1 sec
Speed-2, -4.16, 1000    ; cosine of 2 * 10 rev/s for 1 sec
Wait 1000;
...
```

The programmer has to manually compute the values for the sine and cosine functions at each time step of the simulation. They would also need to manually figure out when the "speed of disc 1 and disc 2 are nearly equal". This is very tedious and error prone which is why it is not done very often.

12.2 SPINNER - Concurrent execution model

SPINNER provides an elegant solution to this problem by supporting concurrent branch execution using the || operator and synchronization using the waitUntil function.

12.2.1 Concurrent branch execution

The concurrent branch operator || allows the user to associate the behavior of the different discs with different parallel blocks of code. Coupled with other nice SPINNER features such as function calls and the ability to query the running simulation properties the solution becomes the following:

```
{
        while(true)
        {
                setSpeed(1, sin(getTime()*10, 1000);
                wait(1000);

                if (getSpeed(2) - getSpeed(1) < 5)
                {
                        break;
                }
        }
} || {
        while(true)
        {
                setSpeed(2, cos(getTime()*10, 1000);
                wait(1000);
                if (getSpeed(2) - getSpeed(1) < 5)
                {
                        break;
                }
        }
}
```

```
        setSpeed(1, 1000);
        setSpeed(2, 1000);
```

In this example, two branches of execution are running in parallel (actually, "lockstep" is the more accurate term). The first branch updates the speed of disc 1 every second based on the sine function. The second one updates disc 2 based on cosine. When their speeds are nearly equal the branches terminate and execution resumes in the main branch.

12.2.2 Comparison of concurrent branch execution (CBE) and threading

Threads and CBE are very similar in that their execution is "interleaved" and the ordering is dependent on the scheduling algorithm. For SPINNER, the scheduling algorithm is round-robin with a 1 instruction "quanta". This means that each branch executes one instruction in a round-robin fashion, from the first branch to the last.

In CBE the parent branch blocks waiting for the child branches to terminate. However, with threads the parent thread continues execution and is scheduled just like the child threads.

The dependency of the parent branch on the child means that the programmer must be careful not to allow a branch to block indefinitely. This is not a problem for threads and, in fact, its very useful for threads to be blocked waiting for input, I/O or an event. This flexibility and power come at a price as multi-threaded programming is notoriously difficult to "get right".

As a programming construct, CBE is declarative whereas multi-threading is dynamic. That is, threading requires the user to allocate and manage thread data structures. SPINNER, on the other hand, implicitly manages the concurrently executing branches. As mentioned before, this simplifies the programming model but constrains the functionality due to the parent-child dependency of the branches.

12.2.3 Synchronization

Often the artist wants to synchronize the behavior of two or more discs. For instance, the artist may want to wait until the angle of disc 1 and 2 are within some amount of each other and then synchronize the speed of disc 1 with that of disc 2. Because SPINscript does not provide any support for this, the artist had to manually calculate how long it would take.

SPINNER provides better support for this functionality with the waitUntil function. With this function, the artist can represent the synchronization condition as a Boolean expression:

```
{
        setSpeed(1, 10, 1000);
        waitUntil(getAngle(2) - getAngle(1) < 5);
        setSpeed(1, getSpeed(2));
        wait(5000);
} || {
        seek(2, 10, 5000);
        wait(5000);
}
```

Chapter 13

References

As SPINNER is implemented in Java, many areas of this reference manual were taken nearly *verbatim* from *The Java Language Specification - Second Edition* by James Gosling, Bill Joy, Guy Steele, and Gilad Bracha.

The C Programming Language, second edition by Brain Kernighhan and Dennis Ritchie was used for many of the lexical conventions of SPINNER.

If something here looks like it exists in the JLS or C LRM, it is safe to assume it is taken from there. (The sincerest form of flattery.)

Part II

Tutorial

Chapter 14

Introduction

This tutorial first describes the basic constructs of the SPIN system: discs. Using these constructs, we describe a first example followed by detail on how to compile these examples. Once the reader is familiar with these basics, we proceed to some advanced constructs.

14.1 The basics

SPINNER spins discs. Each disc has a number of properties associated with it that the programmer can query as the program runs. Discs have simple motion, they rotate. This implies each disc has a speed and position. Speed is a directional quantity with positive speeds indicating clockwise rotation and negative speeds refer to counterclockwise rotation. Speed is measured in revolution per second (rev/sec) and refers to the number of complete rotations (360 degrees) a disc makes in one second. A disc at a standstill has speed of zero. As discs spin, their position, or number of revolutions completed from an initial position of zero, changes. For example, a disc that rotated one full revolution has a position of one. Positive position indicates the number of clockwise rotations a disc has made; negative indicates counterclockwise. Position is maintained as an absolute value throughout the running of the program, and it is reported as a decimal number indicating the whole and fractional amounts of rotation. For example, a disc that has rotated one full revolution and 180 degrees of another will report a position of 1.5.

Discs are referenced by a number: from 1 to the number of discs in the system, inclusive. Discs have additional attributes found in the LRM that relate audio and visual qualities of the disc. Functions for accessing and setting these attributes are found in Chapter 11.

14.1.1 A first example

Here is an example of a very simple SPINNER program:

```
createDiscs(1);
seek(1, 5, 5000);
wait(5000);
```

First, note each statement in the program terminates with a semicolon. The first statement `createDiscs(1);` tells SPINNER to create 1 disc. There can be exactly one `createDisc()` statement in a SPINNER program and it must be called before any other function that acts on a disc is called. Now we have one disc (1). The `seek(1,5,5000);` is one of SPINNER's basic functions for telling a disc how to move (the other is `setSpeed()`). `seek(1,5,5000);` tells disc 1 to rotate to position 5 from it's present position and to get there in 5000 milliseconds (1 sec = 1000 ms. We'll often refer to time constructs in seconds. But note: *all SPINNER functions that require a time parameter expect milliseconds*). It is important to note that commands in SPINNER execute instantaneously. So while we have told disc 1 to seek to position 5, SPINNER proceeds to the next statement before disc 1 has any time to actually rotate. So, what must we do to get disc 1 to actually start seeking? We tell the command processor to pause by issuing a `wait` statement. This effectively stops the command processor from reading lines, for 5 seconds in this case. During this pause, disc 1 is given time to seek per our instructions. After execution of this seek and wait command, disc 1 has speed of zero (it stops after done seeking to a position) and it now has a position of 5.

Another example of a simple SPINNER program:

```
// initialization
if (getDiscsSize()=null) then {
   createDisc(6);
}
double runningTime;
runningTime:=getTime();
seek(1,10,6745);
while(getPosition(1)<=5){
   wait(1);
}
setSpeed(2,3, (getTime()- runningTime));
while(getPostion(2)<=10){
   wait(1);
}
```

Line one is a comment. Comments in SPINNER are defined using `//` or `/* */`. Most `get` functions in SPINNER return literal `null` if the value has not been set or is unknown. Here we use this fact to check if the discs have been initialized using the `getDiscsSize` function (Section 11.24).

Variables are created by defining their type and name. Here, we initialize the value of variable `runningTime` using a system built-in function `getTime()` which return the current running simulation time, in milliseconds.

Now we see use of the `while` loop. `getPosition(num)` returns the present position of disc *num* in decimal. The `wait(1);` advances the simulator 1 millisecond before checking the condition in the `while` again. Here, the `while` loop is used to track the time when a condition becomes true. Alternatively, if we know the wait time needed for disc motion to progress to where the programmer wishes an explicit `wait(time)` could be used.

Also, note the use of the relational operator `<=` in a conditional. A disc's position and speed are stored using double precision numbers. It is recommended that when checking for particular speeds or positions the relational operator is used (as opposed to the equality operator, which is absolute) to ensure that you have not missed your condition by a marginal decimal amount.

Finally, we now see the use of SPINNER's other motion control function `setSpeed`. `setSpeed` takes a disc number, a target speed (rev/sec) and a ramp time (ms) as parameters and directs the appropriate disk to accelerate to target speed in the allotted ramp time. Unlike `seek`, once the disc reaches the target speed it continues to spin until it is directed to do something else.

14.1.2 Compiling and Running

To compile a SPINNER program, you must be sure to have the `spinner.jar` in your local directory and `antlr` libraries must be on your classpath. When this is setup, run

```
% ./spinner.sh <spinner program> <output file>}
compiling...
Done.
```

If there were compilation error, they are displayed to std.out.

14.1.3 Functions example - getAngle()

SPINNER supports the development of user defined functions. Using this feature, it is conceivable that many additional features can be built into the language - from conversion functions to motion procedures. Here, we provide an example that is included in the feature library (more about libraries in Section 14.2.2) which returns the angle of particular disc, calculated in degrees from 0 to 359. This is useful for determining when two discs are in the same orientation. The `getPostion()` built-in function is typically not useful for this type of comparison since it returns the absolute orientation from home position zero. For example, if disc 1 is in position 1.25 and disc 2 is in position 3.25, they are both at 90 degrees from zero position. Note: in this version of the getAngle function, negative positions will not report as positive angles.

```
//return a discs position in degrees 0-359
double func getAngle(double discNum){
   if (getDiscsSize = null) then{ //discs have not been initialized
      return null;
   }
   else{ // disc initialized...ok
      //is discNum reference outside of disc bounds?
      if ((discNum < 0) OR (discNum > getDiscsSize()) then{
         return null;
      }
      else { //discNum boundary ok
         return (getPostion(discNum)%1)*360;
      }
   }
}
```

This function begins with a declaration that identifies it will return a `double` value. The keyword `func` tells the compiler this is a function declaration. `getAngle` is the name the function takes and it requires one parameter, a `double`. After the function declaration is the function logic for calculating the angle of the disc given it's position. The keyword `null` is used as a return type when an error occurs in the function (no discs initialized, disc number parameter is outside the valid disc number bounds, etc.).

To use this function, we must declare it before it's first use in the program. Once declared, it can be used just like any other function. For example:

```
\\assume getAngle was declared
```

```
if (not(getAngle(1)==null)) then {
   if (getAngle(1)>=90) then {
      seek(2, getPostion(1), 0);
   }
}
```

Functions can be called by others functions also; as long as the declaration rules hold.

14.2 Advanced topics

14.2.1 Concurrency

One of SPINNER's key features is its ability to concurrently process *branches* (see Chapter 12 for more details on concurrency details). Here we give a few examples of concurrency.

Let say you want disc 1 to spin to position 5 in 5 seconds. Three seconds after disc one starts spinning, disc 2 should seek to position -7 in 7.5 seconds. This first example simply executes two seek commands concurrently, one 3 seconds after the other:

```
{  //branch 1
   seek(1, 5, 5000);
   wait(5000);
}
||
{  //branch 2
   wait(3000);
   seek(2,-7, 7500);
   wait(7500);
}
seek(3, 1, 1000);
wait(1000);
```

SPINNER executes each branch until it hits a wait state or completes that branch completely. In our example, branch 1 (B1) starts disc 1 seeking and then waits 5 seconds to continue. Branch 2 (B2) is waiting 3 seconds before it proceeds, during which time disc one is seeking. After the 3 seconds wait completes, disc one will continue to seek for 2 more seconds while disc 2 starts spinning. After 2 more seconds, B1 continues executing statements, but there are no more to execute so it completes. B2 waits for its remaining 5.5 seconds before it finishes waiting, continues execution, then ends as there are no more commands to process. Finally, after all the branches have run to completion, the final `seek` is executed; SPINNER waits for 1 second and the program ends.

This was a rather simple example of using concurrency to drive the action of a disc and maintain a set of actions which all must run to completion before the programs continues. More complex examples show how to use concurrency to allow one disc to control the motion of another.

Let us say we want disc 1 to spin using a complicated motion description function we've written. We have no idea when disc 1 will be at position 2, but at the instant it is, we want disc 3 to rotate using the same complicated motion function and to use the speed of disc 1 as it's initial parameter. This is a rather easy thing to do in SPINNER as the following example illustrates:

```
{
   //branch 1
   //call function crazySpin with parameters of disc 1 and speed 0
   //we assume crazySpin runs for some duration of time
   crazySpin(1, 0);
}
||
{
   //branch 2
   waitUntil(getPosition(1)=2);
   crazySpin(3, getSpeed(1));
}
```

We now see the first use of the `waitUntil` built-in function. `waitUntil` takes a condition and pauses execution just as a `wait` statement does. Instead of taking a fixed time value, `waitUntil` pauses until it's condition has a value of true. Think of `waitUntil` as exactly

```
while(not condition){
   wait(1);
}
```

`waitUntil` is a powerful function in SPINNER. It means that you as a programmer need not calculate when an particular condition will occur. Simply state the condition you are waiting for in a `waitUntil` and let SPINNER do the rest.

Here another example of concurrency and waitUntil controlling features using a signal variable:

```
boolean aSignal;
aSignal:= false;
{
   //branch 1
   waitUntil(aSignal); //true after
   seek(2,10,2500);
   waitUntil(getSpeed(2)=0); //true after seek complete
   aSignal:=false;
}
||
{
   //branch 2
   //assume function Teeter returns control after motion started
   TeeterDisc(1, 0);
   waitUntil(getSpeed(1)>=10);
   aSignal:=true;
   waitUntil(not aSignal);
   setSpeed(1,0,3000);
   waitUntil(getSpeed(1)>=0);
}
```

Let walk through what is going on here. First, a boolean variable `aSignal` is defined and set to `false`. We then create two concurrent branches. Branch 1 (B1) immediately goes into a wait state as the signal variable is false and the waitUntil is waiting for it to become true. Branch 2 (B2) now executes `Teeter` and goes into a wait until the speed of disc 1 reaches 1 rev/sec. Once it reaches that speed, it changes the value of `aSignal` to `true` and immediately goes until a wait until `aSignal` return to false. Now that `aSignal` is true, B1 releases it's wait and disc 2 begins a `seek` to position 10. Now B1 goes into a wait state until the speed of disc 2 reaches 0 (this happens when the seek completes). Both branches are in a wait state until either the speed of disc 2 reaches 0 (B1) or the signal variables changes to false (B2). After disc 2's seek is complete, it continues B1 execution, changes the signal variable to false and then ends. Once B1 completes, B2 resumes execution. Since the signal variable changes value to false, B2 resumes from its wait, ramps the speed of disc 1 to 0 in 3 secs, then completes.

As we have seen, the power of function declaration, concurrency, and waitUntil gives SPINNER a level of expressiveness that was quite difficult if not impossible to achieve with SPINscript.

14.2.2 Creating libraries

SPINNER provides a simple mechanism for managing user defined functions and promoting code reuse. Using the `import` function, you can import files which contain function definitions and use those functions in your SPINNER program just as if they were built-in SPINNER functions.

For example, let's assume you have the functions used above (`getAngle()`, `crazySpin()`, and `Teeter`) in a file called "lib1.spinner":

```
//file lib1.spinner

// Function: getAngle
// return a discs position in degrees 0-359
double func getAngle(double discNum){
   if (getDiscsSize = null) then{ //discs have not been initialized
      return null;
   }
   else{ // disc initialized...ok
      //is discNum reference outside of disc bounds?
      if ((discNum < 0) OR (discNum > getDiscSize()) then{
         return null;
      }
      else { //discNum boundary ok
         return (getPostion(discNum)%1)*360;
      }
   }
}

//function crazySpin
//sets disc into a crazy spin motion using seed as seed variable
void func crazySpin(double aDisc, double aSeed) {
...
}
```

```
//function Teeter
//sets disc teetering motion with thePeak as abs value
void func Teeter(double aDisc, double thePeak) {
...
}
```

Now, to use these functions in your SPINNER program, simply import the file using the `import` built-in and begin using your functions.

```
import("lib1.spinner");

... {some code}

if (getAngle(1)=0) then {
   seek(1,.50,1);
}
else {
   seek(1,0,1);
}
```

Important note: is it very important all of your import statements occur at the beginning of your SPINNER program. Otherwise a compile error will occur. Also note all function library import paths should be relative to your main SPINNER program.

Part III

Project Planning

Chapter 15

Introduction

15.1 Team Roles and Responsibilities

SPINNER team organization consists of one team member taking the role of primary lead for each of front end (lexer, parser, semantic analysis), back end (interpreter, simulator), testing, and documentation. These leads have focus through each phase of development. Throughout the project, the team members whose primary focus is not the present focus at the time, work under the direction of the primary lead whose area is the focus. Additionally, one team member is also designated project lead and is responsible for overall coordination among the team for all components of the system. For SPINNER, the project responsibilities break down as follows:

Front end	Sangho Shin
Back end	Marc Eaddy
Testing	Gaurav Khandpur
Documentation	William Beaver
Team Lead	William Beaver

For each of the major system components, various team members are involved through the numerous aspects of development. Here is the breakdown of who did what:

	Design	Programming	Testing	Documentation
Grammar	Sangho Shin, William Beaver	Sangho Shin, Marc Eaddy	Gaurav Khandpur, Sangho Shin	William Beaver, Sangho Shin
Lexer/Parser	Sangho Shin	Sangho Shin	Gaurav Khandpur	William Beaver, Sangho Shin
Semantic Analyzer	Sangho Shin	Sangho Shin, Marc Eaddy	Gaurav Khandpur	William Beaver, Sangho Shin
Interpreter	Marc Eaddy, Sangho Shin, William Beaver	Marc Eaddy	Gaurav Khandpur, William Beaver, Marc Eaddy	William Beaver, Marc Eaddy
Simulator	Marc Eaddy, William Beaver	William Beaver, Marc Eaddy	Gaurav Khandpur, William Beaver	William Beaver, Marc Eaddy

15.2 Process

The development process for SPINNER was comprised of an initial language design phase followed with parser development, simulator prototyping, then semantic analysis. Interfaces as shown in Chapter 16.1.2 were defined between each component before development. Each component then proceeded roughly independently of one another. Using the project plan and tools for collaboration and communication, component development was coordinated where possible to maximize effectiveness and leverage team member knowledge. Possibly contrary to popular trends, email was a quite effective tool for communication among teammates.

15.3 Project timeline and Log

Section	Time	Primary
Whitepaper	09/23	William
Environment setup		
Collaboration platform (twiki)	10/02	William
CVS Setup	11/3	Marc
Front End		Sangho
High Level Language Design	10/03	All
Grammar	10/06	All
Parser	10/16	Sangho
Semantic Analyzer	11/17	Sangho
Code Final	12/03	Sangho
Back End		Marc
Emulator/Prototype Decision	10/03	Marc
Simulator	11/24	Marc
Interpreter Beta	11/24	Marc
Interpreter Final	12/12	Marc
Testing		Gaurav
Test Cases Front End	11/10	Gaurav
Test Cases Back End	11/24	Gaurav
Documentation		William
Whitepaper	9/23	William
LRM	10/2	w/ Sango
Tutorial	11/10	William
Design - Front End	11/10	w/ Sangho
Design - Back End	11/17	w/ Marc
Test Plan	11/24	w/ Gaurav
Lessons Learned	12/12	William
Project Plan	12/16	William
A Code Dump	12/16	William
Final Paper	12/18	William

15.4 Programming Style Guide

Programming style follows the “Code Conventions for the Java Programming Language” by Sun Microsystems, Inc. As each team member uses an IDE for development, much of the textual and formatting style was an artifact of the tool. Some highlights (in part from the Java Style Guide for Central Washington University [http://www.cwu.edu/g̃ellenbe/javastyle/index.html]):

- Indentation- Use three spaces for indentation to indicate nesting of control structures. Avoid the use of tabs for indentation.
- Line length - try an keep lines to less than 80 chars wide

- File Headers - Use template that provides cvs autofill for filename, file revision, file header, and log messages. Header include creator of file, create date, and description.
- Comments - Use comments to provide overviews or summaries of chunks of code and to provide additional information that is not readily available in the code itself. Comment the details of nontrivial or non obvious design decisions; avoid comments that merely duplicate information that is present in and clear from reading the code.
- Variable Names - Choose meaningful names that describe what the variable is being used for. Avoid generic names like number or temp whose purpose is unclear. Compose variable names using mixed case letters starting with a lower case letter. Use plural names for arrays. For example, use testScores instead of testScore. Exception: for loop counter variables are often named simply i, j, or k, and declared local to the for loop whenever possible.
- Use ALL_UPPER_CASE in your constant names, separating words with the underscore character and avoid using magic numbers.

15.5 System Environment

SPINNER lexer, parser, semantic analyzer and interpreter are developed using ANTLR 2.7.2 which then generates Java source. The simulator and other miscellaneous components are built using Java SDK 1.4_2. In compiled form, SPINNER will run on any platform which supports a Java Runtime 1.4.2 or higher.

15.6 Lessons Learned

There were a number of lessons learned by the team:

- Spend more time in language design than you think you need. Think about implementation issues and make sure you have everything covered.
- Think about testing/debugging in language design. For example, is there anything you forgot that will make the life of a developer using your language easier? Remember, as developers of your language you are also its first users and will have an even greater need for robust error handling.
- Tackle key language features as early as possible. Also called risk mitigation. Save easy stuff for later.
- Have unit tests for difficult key systems before you code them. Use the tests as your requirements documentation.
- Implement communication and collaboration tools. CVS and wikiwebs are simple to set up, self documenting, and provide impact immediately. Use your setup time of cvs as a yardstick for progress. For example, if you are a few weeks away from deliverable time and you don't have your development environment established, you should be aware.
- Meet more than once a week. Set milestones for each meeting.
- If you are coding to a target environment, be sure you have access to target environment for reverse engineering as soon as possible.

- Do not let lack of access or knowledge about your target environment bring you to a halt. Make decisions early.
- Make sure target is well specified and understood. Have plan in case aspects of your target will not be well known.

Part IV

System Design

Chapter 16

Introduction

The SPINNER compiler is built using a number of components which are assembled by a few programs to read and parse SPINNER programs into SPINscript. The top level program class structure is identified in Figure 16.1. SPINNER complier is a multi-pass compiler, meaning that the program is "walked" many times depending on the component that is operating on it. While this does slow compilation of SPINNER programs, the speed degradation is negligible relative the general inefficiency of simulating the SPIN environment and a satisfactory trade-off as it reduces complexity and communication needs. This allows developers to operate fairly autonomously while minimizing the risk inherent projects of this type.

As Figure 16.1 indicates, `Main.java` program is the driver for SPINNER. It takes a SPINNER input file and SPINscript output file as parameters. `Main` then instantiates the Preprocessor with the input file. The Preprocessor returns to Main abstract syntax trees for each import file and one for the main SPINNER program. Main then creates a master SpinnerAST from the results of the Preprocessor and invokes Semantic Analyzer on this master SpinnerAST. This creates an Environment object which is passed, along with the master SpinnerAST to the Interpreter for processing. The Interpreter interprets the SpinnerAST using the Simulator, which as the name implies, simulates the effects of the SPINNER program on SPIN. The Simulator generates the final SPINscript code as an artifact of the simulation.

16.1 System Overview

We now discuss the individual components of the SPINNER compiler and the interfaces between them.

16.1.1 Components

Preprocessor

SPINNER implements a preprocessor to handle the `import` built-in function. The preprocessor component reads the main SPINNER program, finds and imports files, and actually uses the lexer and parser (see below) to parse the files into an extension of the ANTLR abstract syntax tree (AST): SpinnerAST. This SpinnerAST is extended from the standard ANTLR AST to support source filename and line number tracking. Due to this, SPINNER correctly reports filename and line numbers in error messages relative to their source file location. The Preprocessor component then returns a number of SpinnerASTs proportional to the number

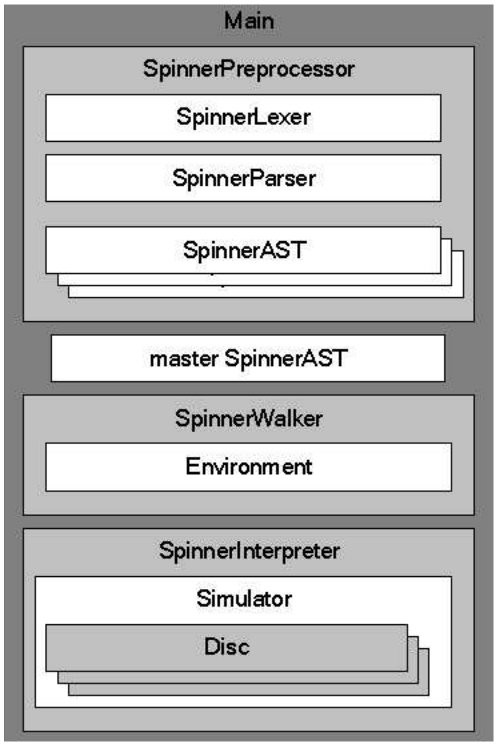

Figure 16.1: SPINNER program structure

of files parsed (original SPINNER program plus any imports) back to Main. Main then combines these multiple SpinnerASTs into one master SpinnerAST before proceeding to Semantic Analysis.

Lexer and Parser

SPINNER's Lexer and Parser are implemented using ANTLR according to the grammar defined in Appendix A.

Semantic Analyzer

The Semantic Analyzer (SpinnerWalker) uses the master SpinnerAST generated by Main via the Preprocessor to, as the name implies, walk the SpinnerAST and perform a semantic check of each node. Semantic analysis does type checking, scope validation, etc. A complete set of error messages generated is found in Appendix C.

When semantic analyzer checks symbols and their types, it creates symbol tables though Environment. First, global symbol table is created, and when it meets new scope it creates new symbol table which is linked to the main symbol table. When a function definition is reached, the semantic analyzer also creates new symbol table, and it is added to the SpinnerFunction object so that it can be used in the interpreter later.

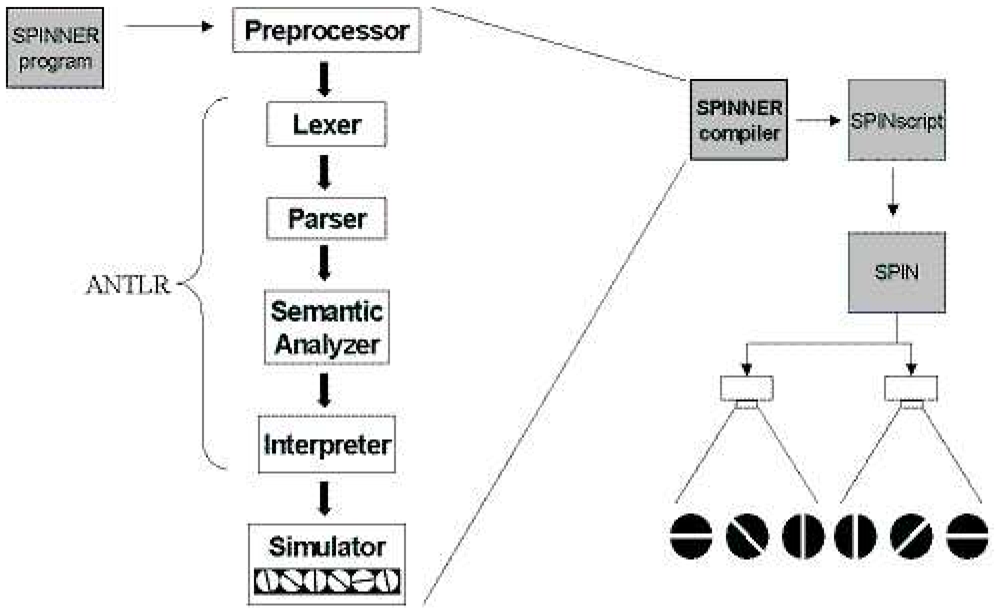

Figure 16.2: SPIN component diagram

Interpreter

SPINNER's Interpreter is specified using ANTLR. The Interpreter walks the master SpinnerAST and actually interprets (executes) each line of Spinner code. The Interpreter uses the Simulator extensively to implement the built-in function calls that query and update the state of the discs.

The Interpreter is fairly standard except for how it implements concurrent branch execution using the parallel operator. When interpreting a parallel branch, the Interpreter goes into a "parallel mode" that affects the behavior of the wait and waitUntil SPINNER builtin functions.

Normally when a wait is encountered, the Interpreter tells the Simulator to wait the supplied amount of time. When a waitUntil is encountered the Interpreter loops waiting for one millisecond until the loop condition is ¡true.

When in "parallel mode", the Interpreter executes each parallel branch in order until the branch completes or hits a wait or waitUntil statement. It then goes on to the next parallel branch, and so on, until all branches have either completed or are waiting (paused). If one or more branches are paused, the Interpreter waits for one millisecond and then resumes each branch at the point that it is paused. This can cause the branch to remain paused waiting for its wait condition. So if two branches have a `wait(2000);` statement, the Interpreter will pause the branches, wait 1 millisecond, then resume the branches, 2000 times. The total amount of time waited will be 2000 milliseconds. Note: If these two branches were executed sequentially instead of concurrently the total wait time would be 4000 milliseconds.

Our SpinnerAST node class includes properties used to implement pausing and resuming execution of expressions. The node stores its local variables with current values using a private symbol table. The node also holds the next "instruction pointer" node which is used to resume execution in the middle of a paused expression. Finally, when resuming an expression, child expressions that have already been executed simply return their already calculated expression value, stored as another property, instead of executing again (which would cause unwanted side-effects).

The combination of local variable symbol tables and stored return values is equivalent to the *state* of the stack whereas the next *instruction pointer* is equivalent to the current stack instruction pointer.

Simulator

The Simulator is used by the Interpreter and mimics the effects of SPINNER programs on SPIN. The Simulator tracks SPIN system settings like gain, volume, etc that are set via a number of setter and getter functions. A Simulator also creates a number of discs as defined in the SPINNER program. SPINNER functions that act on a specific disc are set to the appropriate disc in the simulator and the disc maintains its state as the simulator progresses.

The Simulator advances system time based on the Interpreter's needs. As time advances, the Simulator calls the `play()` method of each disk which advances time and forces the discs to advance their state.

Additionally, as the Interpreter instructs the Simulator to advance, change disc or system state, etc, the Simulator outputs SPINscript appropriately to cause SPIN act the same way the Simulator does. The result is a SPINscript program that instructs SPIN to behave as the SPINNER program indicates.

16.1.2 Interfaces

The Interfaces between the various SPINNER compiler components are simply the SpinnerAST and the Environment.

SpinnerAST

SpinnerAST is an extension of the ANTLR AST. It adds data to nodes allowing the storage and tracking of source file information and line number used in debugging. The SpinnerAST is first generated within the Preprocessor; actually many SpinnerASTs are potentially generated by the preprocessor, one for each imported library file and one for the original SPINNER program. These SpinnerASTs are merged into one master SpinnerAST by Main and are subsequently used by the Semantic Analyzer (SpinnerWalker) and the Interpreter.

Environment

The Environment is an artifact of the Semantic Analyzer. It contains the main symbol table, built-in function table, and methods to track scope, add variables, etc. The Semantic Analyzer uses the Environment for its numerous checks. Certain elements of the Environment are retained after semantic analysis and passed to the Interpreter for use. For example, the built-in function table is maintained and used by the Semantic Analyzer so the Interpreter does not have to reinterpret these functions.

Part V

Testing

Chapter 17

Testing

17.1 Methodology

SPINNER testing takes many forms. In all cases, the compiler is tested using SPINNER programs themselves. These programs vary in complexity as the focus of the components tested change.

Initially, very simple tests were created to exercise the lexer and parser for compliance with the LRM. Once these tests passed, they were expanded to test aspects of the Semantic Analyzer. Finally, these tests were used against the Interpreter checking for expression evaluation. These base tests did not test Interpreter and Simulator interaction.

The next set of tests were written to test built-in functions. As with the earlier Parser tests, the built-in tests exercised the semantic analyzer and interpreters ability to recognize negative cases as well as to accept the positive ones. As the front-end tests passed, the Interpreter began exercising the Simulator more rigorously.

Simulator testing's goal is to ensure that the Simulator is simulating the target environment correctly and that the SPINscript that is output is correct. To achieve the former, a number of programs were written to test disc motion, disc independence, boundary conditions and disc interaction. For the latter, expected SPINscript is hand compiled and manually compared to the output.

Finally, sample SPINscripts were "reverse engineered" into SPINNER program, then compiled, and the final output was manually compared against the original and it was tested in the target environment.

For a complete listing of all test cases, see Appendix E.

17.2 "Automated" Testing Method

Because SPINNER does not support string data types and provides no mechanism for the output of string data, a method was devised to allow automatic verification test results without manual inspection of each case. To accomplish this, many test are structured as follows:

```
//test number 1  =getSpeed on stopped disc
wait(1);
if (getSpeed(1)=0) then { //pass
   setGain(1,1);
} else { //fail
   setGain(1,0);
}
```

The comment indicates the test number and a description of the test. Then a `wait` statement is output with time that matches the test number. Then the test executes. If test passes, then the test outputs a new `gain` value for disc 1 of 1. If the test fails `gain 0.0)` is output. You simply run the test suite which is a series of these cases. When the test completes, search the output file for `gain 0.0` to see if a test failed. If this is not found, then all the tests passed. If a test fails, you can look prior to the `gain 0.0` for the `wait` statement that tells the test number so you know where in the test suite to find the failed test.

17.3 Front-end tests

The front-end tests were written to exercise all portions of the front-end incrementally. These tested both positive and negative conditions. An example of this type of test is:

```
//testing import;
import ("abc.spin");

//testing comments

/* This is a comment.
*/

//this is also a comment

//testing constant names
double MAX_VALUE:=10;

//testing variable and assignments
double a4rfre332dsds;
boolean we323:=true;
boolean we324:=false;
double a:=null;
...
```

See Section E.1.1 for positive tests and Section E.1.2 for negative condition tests.

17.4 Interpreter Testing

Interpreter tests build on the front-end tests. In addition to testing the compilers ability to evaluate expressions, they test many built-in functions. For example:

```
createDiscs(5);
setFile(1,"Aeolean-Harp");  //1,1-readFile Aeolean-Harp;
```

```
seek(all,5,5000);  //1,all-line 5.0 5000;
wait(5000);  //1,wait 5000;
seek(all,0,2500);  //1,all-line 0.0 2500;
;
//asokfja
```

This tests a series of simple built-in functions.

17.5 Simulator Testing

Simulator testing is done in two phases. First, tests of the format above are written to ensure that disc motion is simulated correctly. Boundary conditions are tested, disc interaction is tested, timings and other system settings are also tested. The second phase involves writing a SPINNER program, compiling it manually, and comparing the output to the expected output. The output is also run through SPIN to ensure it behaves as expected.

17.5.1 SPINscript to SPINNER to SPINscript

The final test is to reverse engineer a SPINscript program, write it in SPINNER and compare it's behavior on SPIN with the original SPINscript. An example of a reverse engineered script is found in Section E.4.1. The original SPINscript (cue01) is here:

```
1, all-gain 0.0;
1, --wait-- 25;
1, master-gain 0.3;
1, reverb-send 0.11;
1, reverb-return 0.1;
1, sub-send 0.2;
1, all-slideup 0;
1, all-slidedown 0;
1, all-line 0 0;
1, all-line 0 0;
1, --wait-- 100;
1, all-spinmult 1.;
1, all-sampstart 0.0;
1, all-readfile Aeolean-Harp;
1, --wait-- 8000;
1, all-slideup 10000;
1, all-slidedown 10000;
1, all-gain 0.7;
1, 2-line 4 32000 8 32000 10 16000 12 16000 14 16000;
1, 6-line 4 32000 8 32000 10 16000 12 16000 14 16000;
1, --wait-- 2000;
1, 4-line 4 32000 8 30203.980469 10 16000 12 15102 14 16000;
1, --wait-- 66000;
1, 1-line -2 32000 -10 34000;
```

```
1, 3-line -2.5 40000 -10.5 32000;
1, 5-line -3 48000 -15 32000 -15.5 8000;
1, --wait-- 800;
1, all-slideup 1000;
1, all-slidedown 1000;
1, --wait-- 88000;
5, --end-- 999;
```

And the SPINscript generated from Section E.4.1's testCue01.spinner:

```
1, --------------- Time: 0;
1, all-gain 0.0;
1, --wait-- 25;
1, --------------- Time: 25;
1, master-gain 0.3;
1, reverb-send 0.11;
1, reverb-return 0.1;
1, sub-send 0.2;
1, all-slideup 0.0;
1, all-slidedown 0.0;
1, 1-speed 0.0 0.0;
1, 2-speed 0.0 0.0;
1, 3-speed 0.0 0.0;
1, 4-speed 0.0 0.0;
1, 5-speed 0.0 0.0;
1, 6-speed 0.0 0.0;
1, all-line 0.0 0;
1, --wait-- 100;
1, --------------- Time: 125;
1, all-spinmult 1.0;
1, all-sampstart 0.0;
1, all-readfile Aeolean-Harp;
1, --wait-- 8000;
1, --------------- Time: 8125;
1, all-slideup 10000.0;
1, all-slidedown 10000.0;
1, all-gain 0.7;
1, 2-speed 0.0 0.0;
1, 2-line 4.0 32000;
1, --wait-- 2000;
1, --------------- Time: 10125;
1, 4-speed 0.0 0.0;
1, 4-line 4.0 32000;
1, 6-speed 0.0 0.0;
1, 6-line 4.0 32000;
1, --wait-- 30000;
```

```
1, --------------- Time: 40125;
1, 2-speed 0.0 0.0;
1, 2-line 8.0 32000;
1, --wait-- 2000;
1, --------------- Time: 42125;
1, 4-speed 0.0 0.0;
1, 4-line 8.0 30204;
1, 6-speed 0.0 0.0;
1, 6-line 8.0 30204;
1, --wait-- 30000;
1, --------------- Time: 72125;
1, 2-line 10.0 16000;
1, --wait-- 204;
1, --------------- Time: 72329;
1, 4-speed 0.0 0.0;
1, 4-line 10.0 16000;
1, 6-speed 0.0 0.0;
1, 6-line 10.0 16000;
1, --wait-- 3796;
1, --------------- Time: 76125;
1, 1-speed 0.0 0.0;
1, 1-line -2.0 32000;
1, 3-speed 0.0 0.0;
1, 3-line -2.5 40000;
1, 5-speed 0.0 0.0;
1, 5-line -3.0 48000;
1, --wait-- 12000;
1, --------------- Time: 88125;
1, 2-speed 0.0 0.0;
1, 2-line 12.0 16000;
1, --wait-- 204;
1, --------------- Time: 88329;
1, 4-speed 0.0 0.0;
1, 4-line 12.0 15102;
1, 6-speed 0.0 0.0;
1, 6-line 12.0 15102;
1, --wait-- 15102;
1, --------------- Time: 103431;
1, 4-speed 0.0 0.0;
1, 4-line 14.0 16000;
1, 6-speed 0.0 0.0;
1, 6-line 14.0 16000;
1, --wait-- 694;
1, --------------- Time: 104125;
1, 2-speed 0.0 0.0;
1, 2-line 14.0 16000;
```

```
1, --wait-- 4000;
1, --------------- Time: 108125;
1, 1-speed 0.0 0.0;
1, 1-line -10.0 34000;
1, --wait-- 8000;
1, --------------- Time: 116125;
1, 3-line -10.5 32000;
1, --wait-- 8000;
1, --------------- Time: 124125;
1, 5-line -15.0 32000;
1, --wait-- 32000;
1, --------------- Time: 156125;
1, 5-line -15.5 8000;
1, --wait-- 8000;
1, --------------- Time: 164125;
1, --end-- 999;
```

Part VI

Appendices

Appendix A

The Grammar

```
main_statement
        :outer_statement
        |func_definition
;

outer_statement
        :statement
        |declaration SEMI
        |parallel_statement
;

statement
        :if_statement
        |for_statement
        |while_statement
        |repeat_statement
        |assignment SEMI!
        |func_call SEMI!
        |create_disk SEMI!
        |SEMI!
;

statement_for_func
        :statement
        |return_stmt SEMI
        |declaration SEMI
        |parallel_statement
;

return_stmt
        : "return" (expr)?
```

```
;

inner_statement
        :statement
        |break_statement
        |LBRACE! (main_statement)* RBRACE!
;

break_statement
        : "break" SEMI!
;

if_statement
        : "if" LPAREN! expr RPAREN! "then"! inner_statement
        (options {greedy=true;} : "else"! inner_statement)?
;

for_statement
        : "for" LPAREN! (for_init)? SEMI! (expr)? SEMI! (for_increment)? RPAREN! inner_statement
;

for_init
        :assignment
        |declaration
;

for_increment
        :assignment
;

while_statement
        : "while" LPAREN! expr RPAREN! "do" inner_statement
;

repeat_statement
        : "repeat" inner_statement "until" LPAREN! expr RPAREN!
;

parallel_statement
        : LBRACE! (outer_statement)* RBRACE! (PARALLEL! LBRACE! (outer_statement)* RBRACE!)+
;

create_disk
        : "createDisc"! LBRAKET! NUMBER RBRAKET
;
```

```
declaration
        : variable_declaration
;

variable_declaration
        : type (ID | assignment_with_decl | array_decl)
;

type
        : "double"
        | "boolean"
        | "void"
;

func_definition
        : type "func" ID LPAREN! (args)? RPAREN! LBRACE! func_body RBRACE!
;

func_body
        : (statement_for_func)*
;

args
        : arg (COMMA! arg)*
;

arg
        : type ID
;

func_call
        : ID LPAREN (params)? RPAREN
;

params
        : expr (COMMA! expr)*
;

assignment_with_decl
        :ID ASSIGN! expr
        |auto_inc_assgn
;

assignment
        :assignment_with_decl
        |array ASSIGN expr
```

```
;

auto_inc_assgn
        :ID INC
        |ID DEC
        |INC ID
        |DEC ID
;

expr
        :logic_expr1 ( "or" logic_expr1)*
;

logic_expr1
        :logic_expr2 ("and" logic_expr2)*
;

logic_expr2
        :("not")? relational_expr
;

relational_expr
        :arithmatic_expr1 ((GT | LT | GE | LE | EQ | NE ) arithmatic_expr1)?
;

arithmatic_expr1
        :arithmatic_expr2 ((PLUS | MINUS) arithmatic_expr2)*
;

arithmatic_expr2
        :arithmatic_expr3 ((MULT | DIV | MOD) arithmatic_expr3)*
;

arithmatic_expr3
        : (PLUS | MINUS)? rvalue
;

rvalue
        : func_call
        | NUMBER
        | "true"
        | "false"
        | LPAREN! expr RPAREN!
        | "all"
        | "null"
        | ID
        | STRING
        | array
;

array
        : ID LBRAKET! expr RBRAKET!
;

array_decl
        : array (ASSIGN! LBRACE! expr (COMMA! expr)* RBRACE!)?

;
```

Appendix B

Example Programs

B.1 Program 1

```
/*SPINNER Sample program 1
  author:Gaurav
*/

//initializations

setGain(all,.25);
setReverb(send,1.0);
if (not(getReverb(send) = null)) then {
        setSub(send,1.0);
} else {;}

//empty statement
;

//discs creation and setup
//global variable
createDisc[5];

if (getGain(1)=.25) then {
        setFile(1,"mixer.wav");
}

//set the sound file for all other discs as "melody.wav"
for (double i:=2;i<=num;i++) {
        setFile(i,"melody.wav");
}
```

```
if (isFileSet(1)) then {
        setSpinMult(1,.5);
}
if (getSampleStart(1) = null) then {
        setSampleStart(1,3000);
        setLength(1,2000);
}

setSlide(all,up,5000);
setSlide(all,down,5000);

//disc setup complete

//importing functions
import ("disclib1.spin");

/*a small simulation in which all the discs make 5 revolutions
at 1 rev/sec clockwise and then 5 revolutions at 2 rev/sec ccw
we call this simulation as a standard simulation and make it a function
*/

void func standardSimul() {
        seek(all,5,5000);
        wait(5000);
        seek(all,0,2500);
        wait(2500);
        return;
}

//if all discs are at position 0 then run the standard simulation

boolean flag := true;
for (double i:=1;i<=num;i++) {
        if(not(getPostion(i)=0)) then {
                flag:=false;
        }
}

if(flag) then {
        standardSimul();
}

Rtz(all); // a Lib function

//Done with this program. do quick clean-up
cleanup();  //found in disclib1
```

B.2 Program 2

```
/*SPINNER Sample program 2
  author:Gaurav
*/

import("disclib1.spin");

//initializations
setGain("master",.5);
setReverb("return",.5);
setGain(.5);

//discs setup
createDisc[3];
setFile(all,"remix.wav");
setSampleStart(1,1000);
setSampleStart(2,2000);
setSampleStart(3,3000);
setLength(all,2000);
setSpinMult(1,1.0);
setSpinMult(2,2.0);
setSpinMult(3,-2.0);
setSlide(all,up,0);
setSlide(all,down,0);

/*fork processes.
we start of disc 1 at 1 rev/sec and disc 2 at 2 rev/sec clockwise
simultaneously and when when disc 2's postion > 5 we set disc 1's speed = disc 2
*/

{
        seek(1,10,10000);
        waitUntil(getPosition(2) > 5,-1);
        setSpeed(1,getSpeed(2),0);
}
||
{
        seek(2,10,5000);
        wait(5000);
}

Rtz(all);
/*In this function disc x starts rotating clockwie at 1 rev/sec and dics y at 2 rev/sec clockwise and a
*/
```

```
void func simulCall(double x, double y) {
        boolean flag:=false;
        {
                seek(x,10,10000);
                waitUntil(flag,10000);
                setSpeed(x,y,0);
        }
        ||
        {
                seek(y,10,5000);
                wait(2000);
                flag:=true;
                wait(2000);
                setSpeed(y,x,0);
        }
}

//nested if's, concurrent processing and function call;
if (getPostion(1)=0 and getPosition(2)=0) then {
        if(getSpeed(3)=0 or getPostion(3)=0) then {
                {
                        simulCall(1,2);
                }
                ||
                {
                        setSpeed(3,1.0,1.0);
                }
        }
}

Rtz(all);
cleanup();
```

Appendix C

Compiler Error Messages

Error Number	Error Type
101	Double expected <lineNumber>
102	Boolean expected <lineNumber>
103	Type mismatch <lineNumber>// when assigning double to boolean or vice versa
201	variable <VarName> not defined <lineNumber>
202	variable <VarName> not initialized <lineNumber>
203	variable <VarName> is null <lineNumber>
204	variable <VarName> already exists<lineNumber>
301	missing function definition <lineNumber>
302	missing returnType <lineNumber> //when the return type of a function is missing
303	statement not reachable<lineNumber> //when we have something written after the return statement in a function
304	function <funcName> returnType mismatch<lineNumber>
305	function definition<funcName> already exists <lineNumber>
306	function<funcName> is builtIn<lineNumber> //when trying to create a function that is built-in
401	ArrayIndexOutofBound <lineNumber>//when referencing an array out of its limits

Appendix D

Source Code

D.1 Preprocessor

D.1.1 SpinnerPreprocessor

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SpinnerPreprocessor.java,v 1.7 2003/12/17 04:17:04 ss
//
//    File name: SpinnerProcessor.java
//   Created By: Sangho Shin
//   Created On: December 1, 2003
//
//      Purpose: SpinnerPreprocessor
//
//  Description: It handles imported files, and execute lexer and
//                      parser.
//
////////////////////////////////////////////////////////////////////////
/*
 * $Log: SpinnerPreprocessor.java,v $
 * Revision 1.7  2003/12/17 04:17:04  ss2020
 * If there is any error in Lexer or Parser, it stops there.
 *
 * Revision 1.6  2003/12/14 23:55:25  ss2020
 * Fixed infinite loop bug when setting file name in AST.
 *
 * Revision 1.4  2003/12/11 14:57:30  me133
 * o Now uses SpinnerAST instead of CommonAST
 *
 * Revision 1.3  2003/12/10 07:35:32  me133
 * o Added header comment
 *
 */

package spinner.compiler;

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

import antlr.*;

class SpinnerPreprocessor {

    Vector ASTs;
    SpinnerAST ast;
    String fileName;

    SpinnerPreprocessor(String name) {
        if (!(new File(name)).exists()) {
            System.out.println("File '" + name+"' is imported, but cannot find the file.");
            System.exit(0);
        }
        ASTs = new Vector();
        fileName = name;
        preprocess();;
        createAST();
        if (ast != null) {
            ASTs.add(ast);
        }
    }

    private void preprocess() {

        try {
            FileInputStream fis = new FileInputStream(new File(fileName));
            DataInputStream dis = new DataInputStream(fis);
            BufferedReader br = new BufferedReader(new InputStreamReader(fis));
            String line;
            String fileName;
            while((line = br.readLine()) != null) {
                if (line.startsWith("//") || line.startsWith("/*")) {
                    continue;
                }
                else if (line.trim().startsWith("import")) {
                    int start = line.indexOf("\"");
                    int end = line.indexOf("\"", start+1);
                    fileName = line.substring(start+1, end);
                    SpinnerPreprocessor sp = new SpinnerPreprocessor(fileName);
                    ASTs.addAll(sp.getAST());
                    continue;
                }
            }
        } catch(IOException e) {
            e.printStackTrace();
        }

    }

    public void createAST() {
        try {
            System.out.println("Compiling " + fileName + " ...");
            FileInputStream fis = new FileInputStream(new File(fileName));
            DataInputStream dis = new DataInputStream(fis);
            SpinnerLexer lexer = new SpinnerLexer(dis);
            // Run Parser
            SpinnerParser parser = new SpinnerParser(lexer);
parser.setASTNodeClass(SpinnerAST.class.getName());
            parser.file();
            // Get AST from parser.
            ast = (SpinnerAST) parser.getAST();
            if (ast != null) {
                setFileName(ast);
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
    }

    public Vector getAST() {
        return ASTs;
    }

    private void setFileName(SpinnerAST root) {
        root.setFileName(fileName);
        SpinnerAST child = (SpinnerAST)root.getFirstChild();
        SpinnerAST sibling = (SpinnerAST)root.getNextSibling();
        while (sibling != null) {
            setFileNameChild(sibling);
            sibling = (SpinnerAST)sibling.getNextSibling();
        }
        if (child != null)
```

```
            setFileName(child);

    }

    private void setFileNameChild(SpinnerAST root) {
        root.setFileName(fileName);
        SpinnerAST child = (SpinnerAST)root.getFirstChild();
        if (child != null)
            setFileName(child);

    }

}
```

D.2 Parser and Lexer

D.2.1 Spinner.g

```
header
{
package spinner.compiler;
}
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Spinner.g,v 1.21 2003/12/17 13:45:06 me133 Exp $
//
//    File name: Spinner.g
//   Created By: Sangho Shin
//   Created on: 10/28/2003
//
//      Purpose:
//
//  Description:
//
//////////////////////////////////////////////////////////////////////
/**
 * $Log: Spinner.g,v $
 * Revision 1.21  2003/12/17 13:45:06  me133
 * o BRANCHES are now implemented as STATEMENTS.  Only the node name still says "branches" for clarity
 *
 * Revision 1.20  2003/12/11 14:57:13  me133
 * o Now uses SpinnerAST instead of CommonAST
 *
 * Revision 1.19  2003/12/02 05:06:20  me133
 * o Made waitUntil a keyword/special function
 *
 * Revision 1.18  2003/12/01 07:01:37  me133
 * o Added BRANCH token so that Semantic Analyzer can verify parallel branches
 *
 * Revision 1.17  2003/12/01 05:19:03  me133
 * o wait() function is now considered a keyword/token.  Its a specialization of "func_call".  I did thi
 * o Fixed problem where "master" wasn't recognized
 *
 * Revision 1.16  2003/11/30 23:56:57  me133
 * o Now splits parallel statements into separate branches called "branch"
 *
 * Revision 1.15  2003/11/30 21:37:17  me133
 * o Removed createDiscs (again).  Somehow it snuck back in.
```

```
*
* Revision 1.14  2003/11/30 19:39:30  me133
* o Added main() to help me with my testing
* o BRAKET -> BRACKET
* o Added token for RETURN
* o For init no longer has its own AST node.  We just use whatever assignment or declaration node (or e
* o Restricted for_increment an assignment or a function call
*
* Revision 1.13  2003/11/27 03:59:29  ss2020
*
* Added ARRAY AST.
*
* Revision 1.12  2003/11/24 05:17:14  me133
* o Functions can now have 0 args
* o Cosmetic changes
*
* Revision 1.11  2003/11/24 04:02:03  me133
* o Don't want semicolons to appear in AST
*
* Revision 1.10  2003/11/24 03:35:11  me133
* o Made changes to support "for" and "repeat-until"
* o "for" is now more flexible.  You can have any statement for the for_incr part.
*
* Revision 1.9  2003/11/23 23:19:23  me133
* o GreaterThan and LessThan operators were reversed
*
* Revision 1.8  2003/11/23 21:23:18  me133
* o Trivial changes (added space after : and |)
* o Braces no longer appear as nodes for if statements
*
* Revision 1.7  2003/11/23 21:03:59  me133
* o Now handles unary plus/minus
*
* Revision 1.6  2003/11/23 09:45:35  me133
* o createDiscs is no longer a keyword.  It is just another function.
* o createDiscs no longer uses square brackets.  It uses parenthesis just like any other function.
* o Fixed problem where params node was not created if the function is called without parameters.  Need
*
* Revision 1.5  2003/11/18 14:58:05  ss2020
* Added some AST nodes to the grammar.
*
* Revision 1.4  2003/11/10 23:27:53  me133
* o Added token for "all"
*
* Revision 1.3  2003/11/02 10:35:25  me133
* o Changed package to spinner.compiler
```

```
 * o Added tokens to parser
 * o Parser now creates a decent AST
 * o Missing ! for SEMI for declaration
 * o create_disk -> create_discs
 * o createDisc -> createDiscs (now matches LRM)
 */

class SpinnerLexer extends Lexer;
options { testLiterals = false;
          k = 2;
          charVocabulary = '\3'..'\377';
          exportVocab = SpinnerVocab;
        }

{
    public static void main(String[] args) {
            Main.main(args);
    }
}

PLUS: '+';
MINUS:'-';
MULT:'*';
DIV:'/';
MOD: '%';

ASSIGN:':''=';
INC:'+''+';
DEC:'-''-';

EQ: '=';
GT: '>';
LT: '<';
GE: '>''=';
LE: '<''=';
NE: '!''=';

LPAREN : '(' ;
RPAREN : ')' ;
SEMI : ';' ;
LBRACE: '{';
RBRACE: '}';
LBRACKET: '[';
RBRACKET: ']';
PARALLEL: '|''|';
COLON: ':';
```

```
COMMA: ',';

protected LETTER : ('a'..'z' | 'A'..'Z') ;
protected DIGIT : '0'..'9' ;
ID options { testLiterals = true; }
: LETTER (LETTER | DIGIT | '_')* ;
NUMBER : (DIGIT)+ ('.' (DIGIT)+)?
       | '.' (DIGIT)+
;
STRING : '"'! ('"' '"'! | ~('"'))* '"'! ;

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

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

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

class SpinnerParser extends Parser;
options {
    buildAST = true;
    k = 2;
    exportVocab = SpinnerVocab;
    ASTLabelType = spinner.compiler.SpinnerAST;
}

tokens {
    STATEMENT;
    FUNC_CALL;
    PARAMS;
    FOR_INIT;
    FOR_COND;
    FOR_INCR;
    ASSN_W_DECL;
    VAR_DECL;
    ARGS;
    FUNC_BODY;
    UPLUS;
    UMINUS;
    ARRAY;
    RETURN;
    WAIT;
    WAIT_UNTIL;
}

{
    public static void main(String[] args) {
            Main.main(args);
    }
}

file
        : (main_statement)+ EOF!
        { #file = #([STATEMENT,"PROG"], file); }
;

main_statement
        : outer_statement
        | func_definition
;

outer_statement
        : statement
        | declaration SEMI!
        | parallel_statement
;

statement
        : if_statement
        | for_statement
        | while_statement
        | repeat_statement
        | assignment SEMI!
        | func_call SEMI!
        | SEMI!
;

statement_for_func
        : statement
        | return_stmt SEMI!
        | declaration SEMI!
        | parallel_statement
;
```

```
return_stmt
        : "return"^ (expr)?
;

inner_statement
        : statement
        | break_statement
        | LBRACE! (main_statement)* RBRACE!
        { #inner_statement = #([STATEMENT,"INNER_STMT"], inner_statement); }
;

break_statement
        : "break" SEMI!
;

if_statement
        : "if"^ LPAREN! expr RPAREN! "then"! inner_statement
        (options {greedy=true;} : "else"! inner_statement)?
;

for_statement
        : "for"^ LPAREN! for_init SEMI! for_cond SEMI! for_increment RPAREN! inner_statement
;

for_init
        : assignment
        | declaration
        | /*empty*/
;

for_cond
        : expr
        | /*empty*/
        { #for_cond = #([FOR_COND, "for_cond"], for_cond); }
;

for_increment
        : assignment
        | func_call
        | /*empty*/
        { #for_increment = #([FOR_INCR,"for_increment"], for_increment); }
;

while_statement
        : "while"^ LPAREN! expr RPAREN! inner_statement
```

```
;

repeat_statement
        : "repeat"^ inner_statement "until"! LPAREN! expr RPAREN!
;

parallel_statement
        : LBRACE! parallel_body RBRACE! (PARALLEL! LBRACE! parallel_body RBRACE!)+
        { #parallel_statement = #([PARALLEL, "||"], parallel_statement); }
;

parallel_body
        : (outer_statement)*
        { #parallel_body = #([STATEMENT, "branch"], parallel_body); }
;

declaration
        : variable_declaration
;

variable_declaration
        : type (ID | assignment_with_decl | array)
        { #variable_declaration = #([VAR_DECL,"var_decl"], variable_declaration); }
;

type
        : "double"
        | "boolean"
        | "void"
;

args
        : (arg (COMMA! arg)*)?
        { #args = #([ARGS,"args"], args); }
;

arg
        : type ID
;

func_definition
        : type "func"^ ID LPAREN! args RPAREN! LBRACE! func_body RBRACE!
;

func_body
        : (statement_for_func)*
        { #func_body = #([FUNC_BODY,"func_body"], func_body); }
;

func_call
        : "wait" LPAREN! params RPAREN!
        { #func_call = #([WAIT, "func_call"], func_call); }
        | "waitUntil" LPAREN! params RPAREN!
        { #func_call = #([WAIT_UNTIL, "func_call"], func_call); }
        | ID LPAREN! params RPAREN!
        { #func_call = #([FUNC_CALL,"func_call"], func_call); }
;

params
        : (expr (COMMA! expr)*)?
        { #params = #([PARAMS,"params"], params); }
;

assignment_with_decl
        : ID ASSIGN^ expr
        | auto_inc_assgn
;

assignment
        : assignment_with_decl
        | array ASSIGN^ expr
;

auto_inc_assgn
        : ID INC^
        | ID DEC^
        | INC^ ID
        | DEC^ ID
;

expr
        : logic_expr1 ( "or"^ logic_expr1)*
;

logic_expr1
        : logic_expr2 ("and"^ logic_expr2)*
;

logic_expr2
        : ("not"^)? relational_expr
;
```

```
relational_expr
        : arithmatic_expr1 ((GT^ | LT^ | GE^ | LE^ | EQ^ | NE^ ) arithmatic_expr1)?
;

arithmatic_expr1
        : arithmatic_expr2 ((PLUS^ | MINUS^) arithmatic_expr2)*
;

arithmatic_expr2
        : arithmatic_expr3 ((MULT^ | DIV^ | MOD^) arithmatic_expr3)*
;

arithmatic_expr3
        : PLUS! rvalue
        { #arithmatic_expr3 = #([UPLUS,"UPLUS"], arithmatic_expr3); }
        | MINUS! rvalue
        { #arithmatic_expr3 = #([UMINUS,"UMINUS"], arithmatic_expr3); }
        | rvalue
;

rvalue
        : func_call
        | NUMBER
        | "true"
        | "false"
        | LPAREN! expr RPAREN!
        | "all"
        | "master"
        | "null"
        | ID
        | STRING
        | array
;

array
        : ID LBRACKET! expr RBRACKET!
        { #array = #([ARRAY,"array"], array); }
;
```

D.3 Semantic Analyzer

D.3.1 SemAnal.g

```
header
{
package spinner.compiler;
}
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SemAnal.g,v 1.16 2003/12/17 14:05:14 me133 Exp $
//
//    File name: SemAnal.g
//   Created By: Sangho Shin
//   Created on: 11/17/2003
//
//      Purpose:
//
//  Description:
//
//////////////////////////////////////////////////////////////////////
/**
 * $Log: SemAnal.g,v $
 * Revision 1.16  2003/12/17 14:05:14  me133
 * o Needed to convert node to a SpinnerAST node
 *
 * Revision 1.15  2003/12/14 06:02:26  ss2020
 * Changed error handling routine to support file name and line number.
 *
 * Revision 1.13  2003/12/02 05:07:15  me133
 * o Made waitUntil a keyword/special function
 *
 * Revision 1.12  2003/12/02 02:14:04  me133
 * o Now handles passing array values to functions
 * o Now obtains type of an array value correctly
 *
 * Revision 1.11  2003/12/02 00:57:19  me133
 * o Added more type checking to variable declarations
 * o Fixed bug in variable declarations that didn't handle initial values or array size that were expres
 *
 * Revision 1.10  2003/12/01 06:55:38  me133
 * o Now supports scoping for parallel branches
 *
 * Revision 1.9  2003/12/01 06:16:39  me133
 * o Added verifyFuncCall() which is used by "func_call" and "wait" (shared code)
 * o Function definitions have to also add function parameters to the function scope or else the Semant
 *
 * Revision 1.8  2003/11/30 21:28:59  me133
 * o Fixed compile error
 *
 * Revision 1.7  2003/11/30 21:20:58  me133
 * o if, for, and while statements now use scoping
 * o Now supports repeat-until statements
 * o Added verifyOpArgs, verifyOpDoubleArgs, and verifyOpBooleanArgs to provide verbose error informati
 * o Function call now verifies that arguments are Double, Boolean, or String
 * o Now includes the parameter names when registering a function
 * o Now verifies that the type of the returned value matches the function's return type
 * o Now obtains the type of the value returned by the return statement
 * o Now obtains the type of the value returned by a function call
 * o Now handles type of Strings, "all", "master", "null"
 * o Fixed problem with unary plus/minus
 * o Added more type checking for arthimetic and logic operators
 * o Added mexpr.  Its not currently used.
 *
 * Revision 1.4  2003/11/24 00:39:35  ss2020
 *
 * Reflected UMINUS and UPLUS.
 *
 * Sangho,
 *
 * Revision 1.3  2003/11/24 00:03:59  ss2020
 *
 * Added a lot of sematic analysis.
 *
 * Sangho,
 *
 * Revision 1.2  2003/11/23 09:55:55  me133
 * o Cosmetic changes
 * o Added header comment
 * o Added main()
 * o Changed constant names slightly so as not to confuse them with ANTRL tokens
 * o Added "not"
 *
 */

class SpinnerWalker extends TreeParser;

options {
    importVocab = SpinnerVocab;
}
```

```
{
    public static final int nINIT_TYPE                  = 0;
    public static final int nDOUBLE_TYPE                = 1;
    public static final int nBOOLEAN_TYPE               = 2;
    public static final int nASSIGNMENT_TYPE            = 3;
    public static final int nVOID_TYPE                  = 3; // Same as nASSIGNMENT_TYPE?
    public final static int nARRAY_DOUBLE_TYPE          = 4;
    public final static int nARRAY_BOOLEAN_TYPE         = 5;
    public final static int nSTRING_TYPE = 6;
    public final static int nALL_TYPE                   = 7;
    public final static int nMASTER_TYPE                = 8;
    public final static int nNULL_TYPE                  = 9;

    public static final int nPLUS_SIGN = 101;
    public static final int nMINUS_SIGN = 102;

    Environment env = null;

    public void setEnv(Environment env) {
            this.env = env;
    }

    public void verifyOpDoubleArgs(int lhsType, String lhs, int rhsType, String rhs, String strOpLong, S
    {
        verifyOpArgs(nDOUBLE_TYPE, "number", lhsType, lhs, rhsType, rhs, strOpLong, strOpShort, astNode)
    }

    public void verifyOpDoubleArg(int lhsType, String lhs, String strOpLong, String strOpShort, Spinner
    {
        verifyOpArgs(nDOUBLE_TYPE, "number", lhsType, lhs, nINIT_TYPE, null, strOpLong, strOpShort, ast
    }

    public void verifyOpBooleanArgs(int lhsType, String lhs, int rhsType, String rhs, String strOpLong,
    {
        verifyOpArgs(nBOOLEAN_TYPE, "Boolean expression", lhsType, lhs, rhsType, rhs, strOpLong, strOpS
    }

    public void verifyOpBooleanArg(int lhsType, String lhs, String strOpLong, String strOpShort, Spinne
    {
        verifyOpArgs(nBOOLEAN_TYPE, "Boolean expression", lhsType, lhs, nINIT_TYPE, null, strOpLong, st
    }

    public void verifyOpArgs(int requiredType, String strRequiredType,
                int lhsType, String lhs, int rhsType, String rhs, String strOpLong, String strOpSho
    {
```

```
    assert(lhsType != nINIT_TYPE);

    if (lhsType != requiredType && (rhsType != nINIT_TYPE && rhsType != requiredType))
    {
        env.throwError("Type mismatch. Left- and right-hand sides of " + strOpLong + " operation ("
            strOpShort + ") must be " + strRequiredType + "s. Lhs='" + lhs + "' Rhs='" + rhs + "'."
    }
    else if (lhsType != requiredType)
    {
        assert(lhs != null);
        env.throwError("Type mismatch ('" + lhs + "'). Left-hand side of " + strOpLong + " operatio
            strRequiredType + ".", astNode);
    }
    else if (rhsType != nINIT_TYPE && rhsType != requiredType)
    {
        assert(rhs != null);
        env.throwError("Type mismatch ('" + rhs + "'). Right-hand side of " + strOpLong + " operati
            strRequiredType + ".", astNode);
    }
}

public int verifyFuncCall(SpinnerAST funcAST, AST params) throws RecognitionException
{
    int i = 0;
    String func_name = funcAST.toString();
    int numParams = params.getNumberOfChildren();
    params = params.getFirstChild();
    int paramTypes[] = new int[numParams];
    while(params != null) {

        paramTypes[i] = expr(params);

        String strParamName;
        if (params.getType() == ARRAY)
            strParamName = params.getFirstChild().toString();
        else
            strParamName = params.toString();

        if (paramTypes[i] == nVOID_TYPE) {
            String msg = "'void' type is not allowed for argument " + (i+1) +
                " ('" + strParamName + "') when calling function '" +
                func_name + "'.";
            env.throwError(msg, params);
            return nINIT_TYPE;
        }
        else if (paramTypes[i] != nBOOLEAN_TYPE &&
                paramTypes[i] != nDOUBLE_TYPE &&
                paramTypes[i] != nSTRING_TYPE) {
                String msg = "Argument " + (i+1) + " ('" + strParamName +
                    "') when calling function '" + func_name +
                    "' has an invalid type: " + env.paramToString(paramTypes[i]) +
                    ".";
                env.throwError(msg, params);
                return nINIT_TYPE;
            }

            params = params.getNextSibling();
            i++;
        }

        SpinnerFunction func = env.checkFunction(funcAST, paramTypes, true);

        if (func == null) {
            // Error already reported...
            return nINIT_TYPE;
        }

        return func.getRetType();
    }

    public static void main(String[] args) {
            Main.main(args);
    }

}

expr returns [int t]
{
    t = nINIT_TYPE;
    int a=0;
    int b=0;
    int c=0;
}
: #(STATEMENT (a=expr)*)
| #("if" a=if_cond:expr thenp:. (elsep:.)?) {

    if (a != nBOOLEAN_TYPE)
    {
        env.throwError(102, null, if_cond);
        break;
    }
    env.enterScope();

    if (thenp != null)
        expr(#thenp);

    if (elsep != null)
        expr(#elsep);

    env.leaveScope();
}
| #("for" for_init:. for_cond:. for_increment:. for_body:.) {

    env.enterScope();

    if (for_init != null) {
        expr(#for_init);
    }

    if (for_cond != null &&
        expr(#for_cond) != nBOOLEAN_TYPE) {
        env.throwError(102, null, for_cond);
        env.leaveScope();
        break;
    }

    if (for_increment != null) {
        expr(#for_increment);
    }

    if (for_body != null) {
        expr(#for_body);
    }

    env.leaveScope();
}
| #("while" a=while_cond:expr inner_statment:.) {
    if (a != nBOOLEAN_TYPE) env.throwError(102, null, while_cond);
}
| #("repeat" repeat_body:. repeat_cond:.) {
    if (expr(#repeat_cond) != nBOOLEAN_TYPE) env.throwError(102, null, repeat_cond);
}
| #(EQ a=expr b=expr)
{
    t = nBOOLEAN_TYPE;
}
| #(GT a=lhs_gt:expr b=rhs_gt:expr)
```

```
{
    verifyOpDoubleArgs(a, lhs_gt.toString(), b, rhs_gt.toString(), "greater-than", ">", (SpinnerAST)lhs_
    t = nBOOLEAN_TYPE;
}
| #(GE a=lhs_ge:expr b=rhs_ge:expr)
{
    verifyOpDoubleArgs(a, lhs_ge.toString(), b, rhs_ge.toString(), "greater-than-or-equal-to", ">=", (Sp
    t = nBOOLEAN_TYPE;
}
| #(LT a=lhs_lt:expr b=rhs_lt:expr)
{
    verifyOpDoubleArgs(a, lhs_lt.toString(), b, rhs_lt.toString(), "less-than", "<", (SpinnerAST)lhs_lt)
    t = nBOOLEAN_TYPE;
}
| #(LE a=lhs_le:expr b=rhs_le:expr)
{
    verifyOpDoubleArgs(a, lhs_le.toString(), b, rhs_le.toString(), "less-than-or-equal-to", "<=", (Spinn
    t = nBOOLEAN_TYPE;
}
| #(NE a=expr b=expr)
{
    t = nBOOLEAN_TYPE;
}
| #(ASSIGN var_name:. a=var_value1:expr (b=var_value2:expr)?) {
    if (var_name.getType() == ARRAY) {
        var_name = var_name.getFirstChild();
        int array_type = env.nARRAY_DOUBLE_TYPE - env.nDOUBLE_TYPE;
        if (a == nPLUS_SIGN || a == nMINUS_SIGN) {
            b = b + array_type;  /* Change to array type */
        }
        else  {
            a = a + array_type;  /* Change to array type */
        }
    }

    if (a == nPLUS_SIGN) {
        env.varCheck((SpinnerAST)var_name, b);
    }
    else if (a == nMINUS_SIGN) {
        env.varCheck((SpinnerAST)var_name, b);
    }
    else {
        env.varCheck((SpinnerAST)var_name, a);
    }

    t = nASSIGNMENT_TYPE;
```

```
}
| #(WAIT wait_node:"wait" wait_params:.)
{
    t = verifyFuncCall((SpinnerAST)wait_node, wait_params);
}
| #(WAIT_UNTIL "waitUntil" waitUntil_params:.)
{
    t = nDOUBLE_TYPE;
}
| #(FUNC_CALL func_name:ID params:.)
{
   t = verifyFuncCall((SpinnerAST)func_name, params);
}
| #("func" a=expr def_func_name:ID params2:. func_body:.)
{
    env.enterScope();

    int i = 0;
    int ret = env.OK;
    int numParams = params2.getNumberOfChildren()/2;
    params2 = params2.getFirstChild();
    int args[] = new int[numParams];
    String argNames[] = new String[numParams];
    while(params2 != null) {
        int typeArg = expr(#params2);
        if (typeArg == nDOUBLE_TYPE || typeArg == nBOOLEAN_TYPE || typeArg == nSTRING_TYPE) {
            args[i] = typeArg;
        }
        else {
            env.throwError(309, def_func_name.toString());
            ret = env.ERROR;
            break;
        }
        AST paramName = (AST) params2.getNextSibling();
        assert(paramName != null);
        argNames[i] = paramName.getText();
        env.addVariable((SpinnerAST)paramName, nDOUBLE_TYPE, "0.0");
        params2 = paramName.getNextSibling();
        i++;
    }

    // Verify that the return type and the value returned by the return statement(s)
    // match

    int nReturnedType = expr(func_body);
    env.leaveScope();

    if ((nReturnedType == nINIT_TYPE || nReturnedType == nVOID_TYPE) && a != nVOID_TYPE) {
        env.throwError(310, def_func_name.toString(), def_func_name);
        break;
    }

    if (ret == env.OK) {
        env.addFunction(a, def_func_name.toString(), args, argNames,
            (SpinnerAST) func_body);
    }

    t = a;
}
| #("return" (retVal:.)?) {
    if (retVal == null)
        t = nVOID_TYPE;
    else
        t = expr(retVal);
}
| #(VAR_DECL var_type:. decl_body:.)
{
    AST name = null;

    if (decl_body.getNumberOfChildren() == 0)
    {
        name = (AST) decl_body;
    }
    else
    {
        name = (AST) decl_body.getFirstChild();
    }

    assert(name != null);

    String strArraySize = null;
    int nType = nINIT_TYPE;

    if (var_type.getText().equals("double"))
    {
        if (decl_body.getType() == ARRAY)
            nType = nARRAY_DOUBLE_TYPE;
        else
            nType = nDOUBLE_TYPE;
    }
    else if (var_type.getText().equals("boolean"))
```

```
{
    if (decl_body.getType() == ARRAY)
        nType = nARRAY_BOOLEAN_TYPE;
    else
        nType = nBOOLEAN_TYPE;
}
else {
    env.throwError(402, name.toString(), var_type);
    break;
}

if (decl_body.getType() == ARRAY)
{
    AST sizeNode = (AST) name.getNextSibling();
    assert(sizeNode != null);
    int nSizeType = expr(sizeNode);

    // Make sure array size is a number
    if (nSizeType != nDOUBLE_TYPE) {
        env.throwError(404, null, decl_body);
        break;
    }
}
else if (decl_body.getType() == ASSIGN)
{
    AST valNode = (AST) name.getNextSibling();
    assert(valNode != null);
    int nValType = expr(valNode);

    // Make sure assigned value has the same type as the variable
    if (nType != nValType) {
        env.throwError("Type mismatch. Variable '" + name + "' has type " +
            env.paramToString(nType) + " but value is a " +
            env.paramToString(nValType) + ".", decl_body);
        break;
    }
}

if (strArraySize == null)
{
    env.addVariable((SpinnerAST)name, nType, null);
}
else {
    // We set the array size to an arbitrary valid value "1" because
    // the Semantic Analyzer can't know the size since the size can
    // be an expression.
```

```
            env.addVariable((SpinnerAST)name, nType, "1", null);
        }
    }
    | #(PARAMS (a=expr)* )
    | #(FUNC_BODY func_body2:.)
    {
        t = nINIT_TYPE;
        while (func_body2 != null)
        {
            if (!_tokenSet_0.member(func_body2.getType())) {
                break;
            }
            else if (func_body2.getType() == LITERAL_return) {
                t = expr(func_body2);
                break;
            }
            else {
                func_body2 = func_body2.getNextSibling();
            }
        }

        if (t == nINIT_TYPE)
            t = nVOID_TYPE;
    }
    | #(PARALLEL (a=expr)* )
    | #(BRANCH parallel_branch:.)
    {
        env.enterScope();
        t = expr(parallel_branch);
        env.leaveScope();
    }
    | #(FOR_INIT a=expr)
    | #(PLUS a=lhs_plus:expr b=rhs_plus:expr)
    {
        verifyOpDoubleArgs(a, lhs_plus.toString(), b, rhs_plus.toString(), "addition", #PLUS.getText(), (Sp
        t = nDOUBLE_TYPE;
    }
    | #(MINUS a=lhs_minus:expr b=rhs_minus:expr)
    {
        verifyOpDoubleArgs(a, lhs_minus.toString(), b, rhs_minus.toString(), "subtraction", #MINUS.getText(
        t = nDOUBLE_TYPE;
    }
    | #(MULT a=lhs_mult:expr b=rhs_mult:expr)
    {
        verifyOpDoubleArgs(a, lhs_mult.toString(), b, rhs_mult.toString(), "multiplication", #MULT.getText(
        t = nDOUBLE_TYPE;
```

```
}
| #(MOD a=lhs_mod:expr b=rhs_mod:expr)
{
    verifyOpDoubleArgs(a, lhs_mod.toString(), b, rhs_mod.toString(), "modulo", #MOD.getText(), (Spinner
    t = nDOUBLE_TYPE;
}
| #(DIV a=lhs_div:expr b=rhs_div:expr)
{
    verifyOpDoubleArgs(a, lhs_div.toString(), b, rhs_div.toString(), "division", #DIV.getText(), (Spinne
    t = nDOUBLE_TYPE;
}
| #(INC a=lhs_inc:expr)
{
    verifyOpDoubleArg(a, lhs_inc.toString(), "increment", #INC.getText(), (SpinnerAST)lhs_inc);
    t = nDOUBLE_TYPE;
}
| #(DEC a=lhs_dec:expr)
{
    verifyOpDoubleArg(a, lhs_dec.toString(), "decrement", #DEC.getText(), (SpinnerAST)lhs_dec);
    t = nDOUBLE_TYPE;
}
| #("and" a=lhs_and:expr b=rhs_and:expr)
{
    verifyOpBooleanArgs(a, lhs_and.toString(), b, rhs_and.toString(), "logical AND", "and", (SpinnerAST
    t = nBOOLEAN_TYPE;
}
| #("or" a=lhs_or:expr b=rhs_or:expr)
{
    verifyOpBooleanArgs(a, lhs_or.toString(), b, rhs_or.toString(), "logical OR", "or", (SpinnerAST)lhs
    t = nBOOLEAN_TYPE;
}
| #("not" a=lhs_not:expr)
{
    verifyOpBooleanArg(a, lhs_not.toString(), "logical NOT", "not", (SpinnerAST)lhs_not);
    t = nBOOLEAN_TYPE;
}
| #(ARRAY array_name:. a=expr) {
    if (a != nDOUBLE_TYPE) {
        env.throwError("Array index for array '" + array_name + "' must be a " +
            "positive non-zero number: '" + a + "' is a " +
            env.paramToString(a) + "'", array_name);
        break;
    }

    t = env.varCheck((SpinnerAST)array_name);
    // Since we are referencing an array item, we want the type of the
    // item, not the type of the array variable.
    if (t == nARRAY_DOUBLE_TYPE)
        t = nDOUBLE_TYPE;
    else if (t == nARRAY_BOOLEAN_TYPE)
        t = nBOOLEAN_TYPE;
}
| var:ID { t = env.varCheck((SpinnerAST)var); }
| NUMBER { t=nDOUBLE_TYPE; }
| STRING { t=nSTRING_TYPE; }
| "double" { t=nDOUBLE_TYPE; }
| "boolean" { t=nBOOLEAN_TYPE; }
| "void" { t=nVOID_TYPE; }
| "true" { t=nBOOLEAN_TYPE; }
| "false" { t=nBOOLEAN_TYPE; }
| "all" { t = nDOUBLE_TYPE; }
| "master" { t = nDOUBLE_TYPE; }
| "null" { t = nNULL_TYPE; }
| LBRACE { env.enterScope(); }
| RBRACE { env.leaveScope(); }
| #(UPLUS a=expr) { t = a; }
| #(UMINUS a=expr) { t = a; }
;

mexpr returns [int[] args]
{
    args = null;
}
: #(PARAMS params:.)
{
    int i = 0;
    int ret = env.OK;
    int numParams = params.getNumberOfChildren()/2;

    if (numParams == 0)
        return null;

    params = params.getFirstChild();
    args = new int[numParams];
    String argNames[] = new String[numParams];
    while(params != null) {
        int typeArg = expr(#params);
        if (typeArg == nDOUBLE_TYPE || typeArg == nBOOLEAN_TYPE || typeArg == nSTRING_TYPE) {
            args[i] = typeArg;
        }
        else {
            env.throwError(309, null, params);
            ret = env.ERROR;
            break;
        }
        AST paramName = (AST) params.getNextSibling();
        assert(paramName != null);
        argNames[i] = paramName.getText();
        params = paramName.getNextSibling();
        i++;
    }
}
;
```

D.3.2 SpinnerAST

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SpinnerAST.java,v 1.7 2003/12/17 15:54:47 me133 Exp
//
//    File name: SpinnerAST.java
//   Created By: Sangho Shin & Marc Eaddy
//   Created On: December 3, 2003
//
//      Purpose: Extends AST node to add error info and pause/resume
//
//  Description: By default, ANTLR creates CommonAST nodes which are
// pretty simple.  We extend CommonAST by adding file
// and line number info.  This is used when printing
// out error messages.
//
// In addition, the class contains properties so that
// a child can interupt its processing (pause) and resume
// it later.
//
// A child can throw a PausedException while its expression is
// being evaluated (exprHelper()).  PausedExceptions are only
// thrown when we are executing inside a parallel branch.
// Eventually, the InterpreterHelper.invokeParallel() function
// will catch the exception and then use the node in the
// PauseException object to specify the next node to execute
// for all the children in the branch that threw the exception.
//
// When resuming a branch, we have to skip statements that
// already completed excecuting.  For this we keep the
// return value of the statement around in retLast.  When
// we call exprHelper(), the currentNode.nodeNext property
// will tell us if we should be skipped.  If so, we just
// return retVal without executing again.
//
// The class also keeps the symbol table around just in
// case we are paused and then resumed later.
//
// We also keep around some extra simulation time-related
// properties in the special case where SpinnerAST represents a
// wait or waitUntil node,
//
//////////////////////////////////////////////////////////////////////
```

```
/*
 * $Log: SpinnerAST.java,v $
 * Revision 1.7  2003/12/17 15:54:47  me133
 * o Got rid of startTime.  Just reuse waitTime.
 * o Added comments
 * o Added sanity check
 *
 * Revision 1.6  2003/12/17 14:00:48  me133
 * o Now supports resumeable nodes
 *
 * Revision 1.5  2003/12/17 02:22:57  me133
 * o Added support for pausing and resuming parallel statement (not complete yet)
 * o Comments
 *
 * Revision 1.4  2003/12/13 05:43:59  ss2020
 * Added setFileName().
 *
 * Revision 1.3  2003/12/11 14:58:13  me133
 * o Fixed problem where SpinnerAST was calling antlr to throw a ClassCastException
 *
 * Revision 1.2  2003/12/10 07:35:13  me133
 * o Added header comment
 */

package spinner.compiler;

import antlr.CommonAST;
import antlr.Token;

/**
 * Extends AST node to add error info and pause/resume
 */
public class SpinnerAST extends CommonAST
{
// ---------------------------------------------------
// DATA
// ---------------------------------------------------

private int line = 0;
    private int column = 0;
    private String fileName = null;

// Properties needed to pause and resume

/**
 * this nodes local variables
 */
SymbolTable scope = null;

/**
 * If this node has executed, retLast will have its returned value.
 */
SpinnerRet retLast = null;

/**
 * If this node was paused, nodeNext will point to the immediate
 * child that needs to be executed.
 */
SpinnerAST nodeNext = null;

/**
 * Property needed by wait() and waitUntil() function
 */
double waitTime = -1;

boolean bExecutedAtLeastOnce = false;

// ----------------------------------------------------
// CTOR
// ----------------------------------------------------

    public SpinnerAST() {
    }

    public SpinnerAST(Token tok) {
        super(tok); // This calls initialize() below
    }

// ----------------------------------------------------
// METHODS
// ----------------------------------------------------

/**
 * Initializes the file, line, and column.  Called by ANTLR code.
 */
    public void initialize(Token tok) {
        super.initialize(tok);
        line = tok.getLine();
        column = tok.getColumn();
        fileName = tok.getFilename();
    }
```

```
/**
 * Helper function that either executes the expression for this node
 * or doesn't depending on if the node was paused and then resumed.
 * If its the first time executing then we always execute the expression.
 * If the node or parent node was paused and then resumed, we only execute
 * if this node matches the next instruction ptr (current_node.nodeNext).
 * If we don't execute then its assumed that we already executed at least
 * once so we just return the previous return value.
 */
public SpinnerRet exprHelper(SpinnerInterpreter interpreter, SpinnerAST current_node)
throws antlr.RecognitionException, PausedException
{
assert(current_node != null);

if (current_node.nodeNext == null ||
current_node.nodeNext == this ||
current_node.nodeNext == current_node) // this is a little sloppy - ME
{
retLast = interpreter.expr(this);
bExecutedAtLeastOnce = true;
assert(retLast != null);
current_node.clearNextNode();
}
else
{
// We can skip processing because we aren't the next instruction.
// However, its assumed that we've executed at least once.
assert(bExecutedAtLeastOnce);
}

return retLast;
}

/**
 * line number for this node
 */
    public int getLine() {
        return line;
    }

/**
 * column index for this node
 */
    public int getColumn() {
        return column;
    }
```

```
/**
 * sets the filename for this node
 */
    public void setFileName(String name) {
        fileName = name;
    }

/**
 * filename for this node
 */
    public String getFileName() {
        return fileName;
    }

/**
 * true if this node or one of its children matches the search node
 */
public boolean isSelfOrChild(SpinnerAST search) {
if (this == search)
return true;
else
{
SpinnerAST child = (SpinnerAST) getFirstChild();
while (child != null) {

if (child.isSelfOrChild(search)) {
return true;
}

child = (SpinnerAST) child.getNextSibling();
}

return false;
}
}

/**
 * clears the next instruction ptr (nodeNext) for this node and all its
 * children.  This ensures that this node will be exeucted completely, not
 * resumed at some mid point.
 */
public void clearNextNode() {
nodeNext = null;

SpinnerAST child = (SpinnerAST) getFirstChild();
```

```
while (child != null) {

child.clearNextNode();
child = (SpinnerAST) child.getNextSibling();
}
}

/**
 * Sets the next instruction ptr (nodeNext) for this node and all its
 * children.  This esnures that if this node is restarted, execution
 * will start from the next instruction ptr not from the beginning.
 */
public void setNextNode(SpinnerAST next) {

assert(next != null);
//assert(nodeNext == null || nodeNext == next);

if (next == this)
{
// This is a little sloppy - ME
nodeNext = this;
return;
}

SpinnerAST child = (SpinnerAST) getFirstChild();
while (child != null) {

if (child.isSelfOrChild(next)) {
nodeNext = child;
child.setNextNode(next);
break;
}

child = (SpinnerAST) child.getNextSibling();
}
}

/**
 * Clears all the pause/resume properties (scope, return value, nodeNext,
 * waitTime) for this node and its children. This is needed when we are
 * looping over this node.  However, the semantics of this call need to
 * be tightened.  I'm sure its not called everytime it should be and
 * sometimes its probably called when it shouldn't be called.
 */
public void forgetReturnValue() {
scope = null;
```

```
retLast = null;
nodeNext = null;
waitTime = -1;
bExecutedAtLeastOnce = false;

SpinnerAST child = (SpinnerAST) getFirstChild();
while (child != null) {
child.forgetReturnValue();
child = (SpinnerAST) child.getNextSibling();
}
}

/**
 * Causes this node to enter its local scope.  The local scope is created
 * if necessary.
 */
public void enterScope(Environment env) {
if (scope == null)
{
assert(nodeNext == null);
scope = env.createSymbolTable();
}

env.enterScope(scope);
}

/**
 * causes this node to leave its local scope
 */
public void leaveScope(Environment env) {
env.leaveScope();
}

    /**
 * Print out a child-sibling tree in LISP notation.  If the next instruction
 * ptr (nodeNext) is pointing at a child then it is marked with an asterick (*).
 */
    public String toStringList() {
        SpinnerAST t = this;
        String ts = "";
        if (t.getFirstChild() != null) ts += " (";

ts += " ";

if (nodeNext == this)
ts += "*";
ts += this.toString();
        if (t.getFirstChild() != null) {
            ts += ((SpinnerAST)t.getFirstChild()).toStringList();
        }
        if (t.getFirstChild() != null) ts += " )";
        if (t.getNextSibling() != null) {
            ts += ((SpinnerAST)t.getNextSibling()).toStringList();
        }
        return ts;
    }

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
Main.main(args);
}
}
```

D.3.3 SymbolTable.java

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SymbolTable.java,v 1.2 2003/12/14 06:02:56 ss2020 Exp
//
//    File name: SymbolTable.java
//   Created By: Sangho Shiin
//   Created On: November 17, 2003
//
//      Purpose:
//
//  Description:
//
//////////////////////////////////////////////////////////////////////
/*
 * $Log: SymbolTable.java,v $
 * Revision 1.2  2003/12/14 06:02:56  ss2020
 * Changed error handling routine to support file name and line.
 *
 * Revision 1.1  2003/12/11 15:01:56  me133
 * o Moved SymbolTable and VarDef into SymbolTable.java.  I needed to be able to access SymbolTable fro
 *
 */

package spinner.compiler;

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

public class SymbolTable extends HashMap {
    Environment env;
    LinkedList innerSymbolTables;
    SymbolTable outerSymbolTable;
    boolean useSymbolTable;

    SymbolTable(SymbolTable outer, Environment e, boolean use) {
        outerSymbolTable = outer;
        useSymbolTable = use;
        innerSymbolTables = new LinkedList();
        env = e;
    }

    public SymbolTable enterScope(boolean useSymbolTable) {
        SymbolTable inner;
        if (!useSymbolTable) {
            inner = new SymbolTable(this, env, false);
            innerSymbolTables.addLast(inner);
        }
        else {
inner = new SymbolTable(this, env, false);
innerSymbolTables.addLast(inner);
            inner.setUseSymbolTable(true);
        }
        return inner;
    }

    public void setUseSymbolTable(boolean use) {
        useSymbolTable = use;
    }

    public SymbolTable leaveScope() {
assert(outerSymbolTable != null);
return outerSymbolTable;
    }

    public SymbolTable getOuterSymbolTable() {
        return outerSymbolTable;
    }

    public int getType(String varName) {
        VarDef var = getVariable(varName);
        if (var == null) {
            return env.nINIT_TYPE;
        }
        else {
            return var.getType();
        }
    }

    public void setOuterSymbolTable(SymbolTable table) {
        outerSymbolTable = table;
    }

    public void addVariable(String varName, int type, String value) {

        if (get(varName) != null) {
            env.throwError(204, varName);
        }
        else {
```

```
        VarDef var;
        if (type == env.nBOOLEAN_TYPE) {
            boolean boolValue;
            if (value == null) {
                boolValue = false;
            }
            else {
                boolValue = (new Boolean(value)).booleanValue();
            }
            var = new VarDef(varName, type, boolValue);
            env.debug(varName + " is added as BOOLEAN" +
                  " into symbol table.");
        }
        else {
            double doubleValue;
            if (value == null) {
                doubleValue = 0;
            }
            else {
                doubleValue = (new Double(value)).doubleValue();
            }
            var = new VarDef(varName, type, doubleValue);
            env.debug(varName + " is added as DOUBLE" +
                  " into symbol table.");
        }
        put(varName, var);
    }
}

public void addVariable(SpinnerAST varAST, int type, String value) {

    String varName = varAST.toString();
    if (get(varName) != null) {
        env.throwError(204, varName, varAST);
    }
    else {
        VarDef var;
        if (type == env.nBOOLEAN_TYPE) {
            boolean boolValue;
            if (value == null) {
                boolValue = false;
            }
            else {
                boolValue = (new Boolean(value)).booleanValue();
            }
            var = new VarDef(varName, type, boolValue);
```

```
                env.debug(varName + " is added as BOOLEAN" +
                      " into symbol table.");
            }
            else {
                double doubleValue;
                if (value == null) {
                    doubleValue = 0;
                }
                else {
                    doubleValue = (new Double(value)).doubleValue();
                }
                var = new VarDef(varName, type, doubleValue);
                env.debug(varName + " is added as DOUBLE" +
                      " into symbol table.");
            }
            put(varName, var);
        }
    }

    public void addVariable(String varName, int type, String sizeStr,
                            String value) {
if (sizeStr == null) {
env.throwError("Array size is null.");
return;
}

        int size = Integer.parseInt(sizeStr);
if (size <= 0) {
env.throwError("Array size cannot be 0 or negative. '" + size + "'");
return;
}

        VarDef var;
        if (get(varName) != null) {
            env.throwError(204, varName);
        }
        else if (size < 1) {
            env.throwError(401, varName);
        }
        else {
            if (type == env.nARRAY_BOOLEAN_TYPE) {
                boolean boolValue;
                if (value == null) {
                    boolValue = false;
                }
                else {
```

```
                boolValue = (new Boolean(value)).booleanValue();
            }
            var = new VarDef(varName, type, size, boolValue);
            env.debug(varName + " is added as BOOLEAN ARRAY" +
                  " into symbol table.");
        }
        else if (type == env.nARRAY_DOUBLE_TYPE) {
            double doubleValue;
            if (value == null) {
                doubleValue = 0;
            }
            else {
                doubleValue = (new Double(value)).doubleValue();
            }
            var = new VarDef(varName, type, size, doubleValue);
            env.debug(varName + " is added as DOUBLE ARRAY " +
                  " into symbol table.");
        }
        else
            return;

        put(varName, var);
    }
}

public void addVariable(SpinnerAST varAST, int type, String sizeStr,
                        String value) {

    String varName = varAST.toString();

    if (sizeStr == null) {
        env.throwError("Array size is null.", varAST);
        return;
    }

    int size = Integer.parseInt(sizeStr);
    if (size <= 0) {
        env.throwError("Array size cannot be 0 or negative. '" + size +
                      "'", varAST);
        return;
    }

    VarDef var;
    if (get(varName) != null) {
        env.throwError(204, varName, varAST);
    }
    else if (size < 1) {
        env.throwError(401, varName, varAST);
    }
    else {
        if (type == env.nARRAY_BOOLEAN_TYPE) {
            boolean boolValue;
            if (value == null) {
                boolValue = false;
            }
            else {
                boolValue = (new Boolean(value)).booleanValue();
            }
            var = new VarDef(varName, type, size, boolValue);
            env.debug(varName + " is added as BOOLEAN ARRAY" +
                  " into symbol table.");
        }
        else if (type == env.nARRAY_DOUBLE_TYPE) {
            double doubleValue;
            if (value == null) {
                doubleValue = 0;
            }
            else {
                doubleValue = (new Double(value)).doubleValue();
            }
            var = new VarDef(varName, type, size, doubleValue);
            env.debug(varName + " is added as DOUBLE ARRAY " +
                  " into symbol table.");
        }
        else
            return;

        put(varName, var);
    }
}

public int varCheck(String varName, int type) {
    VarDef var = getVariable(varName);
    int ret = 0;
    if (var != null) {
        if (var.getType() != type) {
            env.throwError(103, varName);
            ret = env.ERROR;
        }
        else {
            ret = var.getType();
        }
    }
    else {
        ret = env.ERROR;
    }

    return ret;
}


public int varCheck(SpinnerAST varAST, int type) {
    VarDef var = getVariable(varAST);
    int ret = 0;
    if (var != null) {
        if (var.getType() != type) {
            env.throwError(103, varAST.toString(), varAST);
            ret = env.ERROR;
        }
        else {
            ret = var.getType();
        }
    }
    else {
        ret = env.ERROR;
    }

    return ret;
}

public int varCheck(String varName) {
    VarDef var = getVariable(varName);
    if (var == null) {
        return env.ERROR;
    }
    else {
        return var.getType();
    }
}

public int varCheck(SpinnerAST varAST) {
    VarDef var = getVariable(varAST);
    if (var == null) {
        return env.ERROR;
    }
    else {
        return var.getType();
    }
    }

    public boolean getBooleanValue(String varName) {
        VarDef var = getVariable(varName);
        if (var == null) {
            env.throwError(201, varName);
            return false;
        }
        else if (var.getType() != env.nBOOLEAN_TYPE) {
            env.throwError(103, varName);
            return false;
        }
        else {
            return var.getBooleanValue();
        }
    }

    public boolean getBooleanValue(String varName, int index) {
        VarDef var = getVariable(varName);
        if (var == null) {
            env.throwError(201, varName);
            return false;
        }
        else if (var.getType() != env.nARRAY_BOOLEAN_TYPE) {
            env.throwError(103, varName);
            return false;
        }
        else if (index <= 0 || index > var.getSize()) {
            env.throwError(401, varName);
            return false;
        }
        else {
            return var.getBooleanValue(index);
        }
    }

    public double getDoubleValue(String varName) {
        VarDef var = getVariable(varName);
if (var == null) {
           env.throwError(201, varName);
return -1;
}
        if (var.getType() != env.nDOUBLE_TYPE && var.getType() != env.nBOOLEAN_TYPE) {
            env.throwError(103, varName);
            return -1;
        }
        else {
            return var.getDoubleValue();
        }
    }

    public double getDoubleValue(String varName, int index) {
        VarDef var = getVariable(varName);
        if (var == null) {
            env.throwError(201, varName);
            return -1;
        }
        else if (var.getType() != env.nARRAY_DOUBLE_TYPE) {
            env.throwError(103, varName);
            return -1;
        }
        else if (index <= 0 || index > var.getSize()) {
            env.throwError(401, varName);
            return -1;
        }

        return var.getDoubleValue(index);
    }

    public void setValue(String varName, String value) {
        VarDef var = getVariable(varName);
        if (var == null) {
            env.throwError(201, varName);
        }
        else if (var.getType() == env.nBOOLEAN_TYPE) {
            var.setValue((new Boolean(value)).booleanValue());
        }
        else if (var.getType() == env.nDOUBLE_TYPE) {
            var.setValue((new Double(value)).doubleValue());
        }
else
{
env.debug("setValue() called with an invalid variable type. Name=" + varName + " Type=" + var.getType()
}
    }

    public void setValue(String varName, int nIndex, String value) {
        VarDef var = getVariable(varName);
        if (var == null) {
            env.throwError(201, varName);
        }
        else if (var.getType() == env.nARRAY_BOOLEAN_TYPE) {
            var.setValue((new Boolean(value)).booleanValue(), nIndex);
        }
        else if (var.getType() == env.nARRAY_DOUBLE_TYPE) {
            var.setValue((new Double(value)).doubleValue(), nIndex);
        }
else
{
env.debug("setValue() called with an invalid variable type. Name=" + varName + " Type=" + var.getType()
}
    }

    public VarDef getVariable(String varName) {
        SymbolTable symbolTable = this;
        VarDef var;

        while(symbolTable != null) {
            if ((var = (VarDef)symbolTable.get(varName)) != null) {
                return var;
            }
            symbolTable = symbolTable.getOuterSymbolTable();
        }

        env.throwError(201, varName);
        return null;
    }

    public VarDef getVariable(SpinnerAST varAST) {
        SymbolTable symbolTable = this;
        VarDef var;
        String varName = varAST.toString();

        while(symbolTable != null) {
            if ((var = (VarDef)symbolTable.get(varName)) != null) {
                return var;
            }
            symbolTable = symbolTable.getOuterSymbolTable();
        }

        env.throwError(201, varName, varAST);
        return null;
    }

}

class VarDef {
    int varType;
```

```
String varName;
boolean varBooleanValue;
double varDoubleValue;
boolean arrayBooleanValues[];
double arrayDoubleValues[];
int arraySize;

VarDef(String name, int type, boolean value) {
    varName = name;
    varType = type;
    varBooleanValue = value;
}

VarDef(String name, int type, double value) {
    varName = name;
    varType = type;
    varDoubleValue = value;
}

VarDef(String name, int type, int size, boolean value) {
    varName = name;
    varType = type;
    arraySize = size;
    arrayBooleanValues = new boolean[arraySize];
}

VarDef(String name, int type, int size, double value) {
    varName = name;
    varType = type;
    arraySize = size;
    arrayDoubleValues = new double[arraySize];
}

public boolean getBooleanValue() {
    return varBooleanValue;
}

public double getDoubleValue() {
    return varDoubleValue;
}

public boolean getBooleanValue(int index) {
    return arrayBooleanValues[index-1];
}

public double getDoubleValue(int index) {
        return arrayDoubleValues[index-1];
    }

    public int getType() {
        return varType;
    }

    public void setValue(boolean value) {
        varBooleanValue = value;
    }

    public void setValue(double value) {
        varDoubleValue = value;
    }

    public void setValue(double value, int index) {
        arrayDoubleValues[index-1] = value;
    }

    public void setValue(boolean value, int index) {
        arrayBooleanValues[index-1] = value;
    }

    public int getSize() {
        return arraySize;
    }
}
```

D.4 Interpreter

D.4.1 Interpreter.g

```
header
{
package spinner.compiler;
}
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Interpreter.g,v 1.19 2003/12/17 15:53:38 me133 Exp $
//
//    File name: Interpreter.g
//   Created By: Sangho Shin, Marc Eaddy
//   Created on: 11/17/2003
//
//      Purpose:
//
//  Description:
//
///////////////////////////////////////////////////////////////////////
/**
 * $Log: Interpreter.g,v $
 * Revision 1.19  2003/12/17 15:53:38  me133
 * o Got rid of startTime.  Just reuse waitTime.
 *
 * Revision 1.18  2003/12/17 15:21:47  me133
 * o Added some comments
 *
 * Revision 1.17  2003/12/17 14:06:00  me133
 * o Now supports nested parallel statements and pausing and resuming statements
 *
 * Revision 1.16  2003/12/02 05:10:09  me133
 * o Made waitUntil a keyword/special function
 * o Now uses InterpreterHelper for large Java code segments
 *
 * Revision 1.15  2003/12/02 02:15:38  me133
 * o Now supports arrays
 *
 * Revision 1.14  2003/12/01 07:00:22  me133
 * o Now supports scoping for parallel branches
 * o Small name change
 *
 * Revision 1.13  2003/12/01 06:17:59  me133
 * o Added invokeFunc() which is used by "func_call" and "wait" (shared code)
 * o Implemented support for PARALLEL branch execution using a simple AST walking approach
 *
 * Revision 1.12  2003/11/30 20:06:04  me133
 * o Now handles variables, scope and function parameters
 * o Now uses the SpinnerWalker constants
 * o Added safeCompareTo() for comparing two Double objects that may be null
 * o Now interprets function bodies correctly.  Stops when it hits the first return statement.
 * o Now scopes if, for, repeat-until, and while statements correctly.  Also, these statements do not pr
 * o For loop now works properly
 * o Assignment no longer checks the variable first.  Its assumed the SemAnalyzer did this already.
 * o Now supports ++ and --
 * o Now registers and passes parameters to functions properly, using a new scope.
 * o Now supports the return statement
 * o Now supports boolean declaration+assignment expressions.  Still needs work to handle declarations o
 * o "all" and "master" are keywords
 * o Now returns correct value for "true", "false", and "null"
 *
 * Revision 1.11  2003/11/24 05:17:43  me133
 * o Now handles executing the function body
 *
 * Revision 1.10  2003/11/24 04:43:50  me133
 * o Fixed problems with func definition and calling.  Still doesn't work however.
 *
 * Revision 1.9  2003/11/24 04:03:05  me133
 * o Fixed null ptr when handling func definition
 *
 * Revision 1.8  2003/11/24 03:32:07  me133
 * o Partially implemented "for" (needs to support variables)
 * o Implemented "repeat-until"
 *
 * Revision 1.7  2003/11/23 23:36:02  me133
 * o Implemented "while"
 *
 * Revision 1.6  2003/11/23 23:27:39  me133
 * o Implemented "not"
 *
 * Revision 1.5  2003/11/23 21:57:14  me133
 * o Now handles if-then, if-then-else, and if-then-else-if-then-else statements
 * o Now handles compound boolean expressions (constructed using "and" and "or")
 * o Commented out enterScope() and leaveScope() because it was throwing exceptions
 *
 * Revision 1.4  2003/11/23 21:03:58  me133
 * o Now handles unary plus/minus
 *
 * Revision 1.3  2003/11/23 09:54:39  me133
```

```
 * o Added math operations
 * o Cosmetic changes
 * o Added main() function
 * o Now actually executes builtin functions as they are walked
 * o Started using Environment to obtain functions to invoke
 *
 * Revision 1.2  2003/11/19 22:52:25  me133
 * o Small improvements to AST-based ANTLR grammar.  Probably won't be using this in the end.
 *
 * Revision 1.1  2003/11/17 07:38:36  me133
 * o Initial checkin
 */

class SpinnerInterpreter extends TreeParser;

options {
    importVocab = SpinnerVocab;
    ASTLabelType = spinner.compiler.SpinnerAST;
}

{
    public static final int nINIT_TYPE          = SpinnerWalker.nINIT_TYPE;
    public static final int nDOUBLE_TYPE        = SpinnerWalker.nDOUBLE_TYPE;
    public static final int nBOOLEAN_TYPE       = SpinnerWalker.nBOOLEAN_TYPE;
    public static final int nARRAY_DOUBLE_TYPE  = SpinnerWalker.nARRAY_DOUBLE_TYPE;
    public static final int nARRAY_BOOLEAN_TYPE = SpinnerWalker.nARRAY_BOOLEAN_TYPE;

    Simulator sim;
    Environment env;

    public void setEnvironment(Environment env) {
        this.env = env;
    }

    public void setSimulator(Simulator sim) {
        this.sim = sim;
    }

    public static void main(String[] args) {
        Main.main(args);
    }

}

expr returns [SpinnerRet r] throws PausedException
{
    r = null;
    SpinnerAST current_node = (SpinnerAST) _t;
}
: #(STATEMENT (statement:.)?)
{
    while (statement != null)
    {
        statement.exprHelper(this, current_node);
        statement = (SpinnerAST) statement.getNextSibling();
    }

    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| #("if" if_cond:. thenp:. (elsep:.)?)
{
    if_cond.exprHelper(this, current_node);

    if (if_cond.retLast.isTrue())
    {
        thenp.enterScope(env);
        try {
            thenp.exprHelper(this, current_node);
        }
        finally {
            thenp.leaveScope(env);
        }
    }
    else if (elsep != null)
    {
        elsep.enterScope(env);
        try {
            elsep.exprHelper(this, current_node);
        }
        finally {
            elsep.leaveScope(env);
        }
    }

    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| #("for" for_init:. for_cond:. for_incr:. (for_body:.)?) {

    current_node.enterScope(env);
    try {

        for_init.exprHelper(this, current_node);
        for_cond.exprHelper(this, current_node);

        while (for_cond.retLast.isTrue()) {

            if (for_body != null) // body can be empty
            {
                for_body.exprHelper(this, current_node);

                // Need to forget the return value since we've finished
                // executing the body without any PausedExceptions. We
                // need to do this because we are looping over (revisiting)
                // the for_body node.
                for_body.forgetReturnValue();
            }

            for_incr.exprHelper(this, current_node);

            // Have to forget these for the same reason as above
            for_incr.forgetReturnValue();
            for_cond.forgetReturnValue();

            for_cond.exprHelper(this, current_node);
        }

        r = SpinnerRet.retVOID;
        current_node.retLast = r;
    }
    finally {
        current_node.leaveScope(env);
    }
}
| #("while" while_cond:. (while_body:.)?) {

    current_node.enterScope(env);

    try
    {
        while_cond.exprHelper(this, current_node);

        while (while_cond.retLast.isTrue())
        {
            if (while_body != null)
            {
                while_body.exprHelper(this, current_node);
                // Have to forget these for the same reason as above
                while_body.forgetReturnValue();
            }

            // Have to forget these for the same reason as above
            while_cond.forgetReturnValue();

            while_cond.exprHelper(this, current_node);
        }

        r = SpinnerRet.retVOID;
        current_node.retLast = r;
    }
    finally {
        current_node.leaveScope(env);
    }
}
| #("repeat" repeat_body:. repeat_cond:.) {

    current_node.enterScope(env);

    try {
        do {
            repeat_body.exprHelper(this, current_node);

            // Have to forget these for the same reason as above
            repeat_body.forgetReturnValue();
            repeat_cond.forgetReturnValue();

            repeat_cond.exprHelper(this, current_node);
        }
        while (repeat_cond.retLast.isFalse());

        r = SpinnerRet.retVOID;
        current_node.retLast = r;
    }
    finally {
        current_node.leaveScope(env);
    }
}
| #("and" lhs_and:. rhs_and:.) {

    lhs_and.exprHelper(this, current_node);
    rhs_and.exprHelper(this, current_node);
    r = SpinnerRet.safeAnd(lhs_and.retLast, rhs_and.retLast);
    current_node.retLast = r;
}
| #("or" lhs_or:. rhs_or:.) {

    lhs_or.exprHelper(this, current_node);
    rhs_or.exprHelper(this, current_node);

    r = SpinnerRet.safeOr(lhs_or.retLast, rhs_or.retLast);
    current_node.retLast = r;
}
| #("not" not_expr:.) {

    not_expr.exprHelper(this, current_node);

    r = SpinnerRet.safeNot(not_expr.retLast);
    current_node.retLast = r;
}
| #(PLUS lhs_plus:. rhs_plus:.) {

    lhs_plus.exprHelper(this, current_node);
    rhs_plus.exprHelper(this, current_node);

    r = SpinnerRet.safeAdd(lhs_plus.retLast, rhs_plus.retLast);
    current_node.retLast = r;
}
| #(MINUS lhs_minus:. rhs_minus:.) {

    lhs_minus.exprHelper(this, current_node);
    rhs_minus.exprHelper(this, current_node);

    r = SpinnerRet.safeSubtract(lhs_minus.retLast, rhs_minus.retLast);
    current_node.retLast = r;
}
| #(MULT lhs_mult:. rhs_mult:.) {

    lhs_mult.exprHelper(this, current_node);
    rhs_mult.exprHelper(this, current_node);

    r = SpinnerRet.safeMultiply(lhs_mult.retLast, rhs_mult.retLast);
    current_node.retLast = r;
}
| #(DIV lhs_div:. rhs_div:.) {

    lhs_div.exprHelper(this, current_node);
    rhs_div.exprHelper(this, current_node);
```

```
        r = SpinnerRet.safeDivide(lhs_div.retLast, rhs_div.retLast);
        current_node.retLast = r;
    }
    | #(MOD lhs_mod:. rhs_mod:.) {

        lhs_mod.exprHelper(this, current_node);
        rhs_mod.exprHelper(this, current_node);

        r = SpinnerRet.safeMod(lhs_mod.retLast, rhs_mod.retLast);
        current_node.retLast = r;
    }
    | #(UPLUS uplus_expr:.) {

        uplus_expr.exprHelper(this, current_node);
        r = uplus_expr.retLast;
        current_node.retLast = r;
    }
    | #(UMINUS uminus_expr:.) {

        uminus_expr.exprHelper(this, current_node);
        r = uminus_expr.retLast.switchSign();
        current_node.retLast = r;
    }
    | #(INC varName:ID) {

        int t = env.varCheck(varName.toString());
        if (t != env.ERROR) {
            r = new SpinnerRet(env.getDoubleValue(varName.toString()) + 1);
            env.setValue(varName.toString(), r.toString());
            current_node.retLast = r;
        }
    }
    | #(DEC varName2:ID) {

        int t = env.varCheck(varName2.toString());
        if (t != env.ERROR) {
            r = new SpinnerRet(env.getDoubleValue(varName2.toString()) - 1);
            env.setValue(varName2.toString(), r.toString());
            current_node.retLast = r;
        }
    }
    | #(EQ lhs_eq:. rhs_eq:.) {

        lhs_eq.exprHelper(this, current_node);
        rhs_eq.exprHelper(this, current_node);
        r = SpinnerRet.safeCompareTo(lhs_eq.retLast, rhs_eq.retLast) == 0 ? SpinnerRet.retTRUE : SpinnerRet
        current_node.retLast = r;
    }
    | #(NE lhs_ne:. rhs_ne:.) {

        lhs_ne.exprHelper(this, current_node);
        rhs_ne.exprHelper(this, current_node);

        r = SpinnerRet.safeCompareTo(lhs_ne.retLast, rhs_ne.retLast) != 0 ? SpinnerRet.retTRUE : SpinnerRet
        current_node.retLast = r;
    }
    | #(GT lhs_gt:. rhs_gt:.) {

        lhs_gt.exprHelper(this, current_node);
        rhs_gt.exprHelper(this, current_node);

        r = SpinnerRet.safeCompareTo(lhs_gt.retLast, rhs_gt.retLast) > 0 ? SpinnerRet.retTRUE : SpinnerRet.
        current_node.retLast = r;
    }
    | #(GE lhs_ge:. rhs_ge:.) {

        lhs_ge.exprHelper(this, current_node);
        rhs_ge.exprHelper(this, current_node);

        r = SpinnerRet.safeCompareTo(lhs_ge.retLast, rhs_ge.retLast) >= 0 ? SpinnerRet.retTRUE : SpinnerRet
        current_node.retLast = r;
    }
    | #(LT lhs_lt:. rhs_lt:.) {

        lhs_lt.exprHelper(this, current_node);
        rhs_lt.exprHelper(this, current_node);

        r = SpinnerRet.safeCompareTo(lhs_lt.retLast, rhs_lt.retLast) < 0 ? SpinnerRet.retTRUE : SpinnerRet.
        current_node.retLast = r;
    }
    | #(LE lhs_le:. rhs_le:.) {

        lhs_le.exprHelper(this, current_node);
        rhs_le.exprHelper(this, current_node);

        r = SpinnerRet.safeCompareTo(lhs_le.retLast, rhs_le.retLast) <= 0 ? SpinnerRet.retTRUE : SpinnerRet
        current_node.retLast = r;
    }
    | #(WAIT "wait" wait_params:.)
    {
```

```
    if (!env.isInParallelBranch())
    {
        // Ignore return value
        InterpreterHelper.invokeFunction(env, sim, this, "wait", current_node,
            wait_params);
    }
    else
    {
        double currentSimulationTime = sim.getTime();

        if (current_node.waitTime == -1) {
            // This is the first time wait() has been called.  Calculate
            // the wait time.  Wait time is the current simulation time
            // plus the wait time parameter.
            SpinnerAST wait_param = (SpinnerAST) wait_params.getFirstChild();

            wait_param.exprHelper(this, current_node);
            current_node.waitTime = currentSimulationTime + wait_param.retLast.doubleValue();
        }

        if (currentSimulationTime < current_node.waitTime)
        {
            throw new PausedException(current_node);
        }
    }

    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| #(WAIT_UNTIL "waitUntil" waitUntil_params:.)
{
    if (current_node.waitTime == -1)
    {
        current_node.waitTime = sim.getTime();
    }

    SpinnerAST waitUntilCond = (SpinnerAST) waitUntil_params.getFirstChild();
    SpinnerAST waitUntilTimeout = (SpinnerAST) waitUntilCond.getNextSibling();

    // timeout value is supposed to be there but some early tests forgot
    // to include it.  So we make it optional (hey its overloaded!)
    double timeout = -1;
    if (waitUntilTimeout != null)
    {
        // Have to recalculate the value each time since we are essentially
        // looping over waitUntil
        waitUntilTimeout.forgetReturnValue();
        waitUntilTimeout.exprHelper(this, waitUntil_params);
        timeout = waitUntilTimeout.retLast.doubleValue();
    }

    if (!env.isInParallelBranch()) {

        waitUntilCond.exprHelper(this, current_node);
        while ( waitUntilCond.retLast.isFalse() &&
                (timeout == -1 || sim.getTime() < current_node.waitTime + timeout))
        {
            sim.play(1);

            // Have to recalculate the value each time since we are essentially
            // looping over waitUntil
            waitUntilCond.forgetReturnValue();
            waitUntilCond.exprHelper(this, waitUntil_params);
        }
    }
    else
    {
        // Have to recalculate the value each time since we are essentially
        // looping over waitUntil
        waitUntilCond.forgetReturnValue();
        waitUntilCond.exprHelper(this, waitUntil_params);
        if (waitUntilCond.retLast.isFalse() &&
            (timeout == -1 || sim.getTime() < current_node.waitTime + timeout))
        {
            throw new PausedException(current_node);
        }
    }

    // Return how long it took us
    r = new SpinnerRet(sim.getTime() - current_node.waitTime);
    current_node.retLast = r;
}
| #(FUNC_CALL func_name:ID params:.)
{
    r = InterpreterHelper.invokeFunction(env, sim, this, func_name.toString(),
            current_node, params);
    current_node.retLast = r;
}
| #("return" (retVal:.)?) {
    if (retVal != null)
    {
        retVal.exprHelper(this, current_node);
        r = retVal.retLast;
    }
    else {
        r = SpinnerRet.retVOID;
    }

    current_node.retLast = r;
}
| #("func" type:. def_func_name:ID def_func_params:. def_func_body:.)
{
    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| #(FUNC_BODY func_body:.)
{
    r = SpinnerRet.retVOID;

    current_node.enterScope(env);

    try {
        while (func_body != null)
        {
            // Only the value in the "return" statement is
            // returned
            if (func_body.getType() == LITERAL_return) {
                func_body.exprHelper(this, current_node);
                r = func_body.retLast;
                break;
            }
            else {
                func_body.exprHelper(this, current_node);
                func_body = (SpinnerAST) func_body.getNextSibling();
            }
        }

        current_node.retLast = r;
    }
    finally {
        current_node.leaveScope(env);
    }
}
| #(VAR_DECL var_type:. decl_body:.)
{
    r = InterpreterHelper.declareVariable(env, this, current_node, var_type, decl_body);
    current_node.retLast = r;
}
```

```
| #(ASSIGN var_name:. var_value1:.) {

   var_value1.retLast = expr(#var_value1);

    if (var_name.getType() == ARRAY) {

        SpinnerAST aryName = (SpinnerAST) var_name.getFirstChild();
        assert(aryName != null);

        SpinnerAST arySize = (SpinnerAST) aryName.getNextSibling();
        assert(arySize != null);

        arySize.exprHelper(this, current_node);

        double nIndex = arySize.retLast.doubleValue();

        if (nIndex != (int) nIndex) {
            env.throwError("Array index for array '" + aryName + "' cannot be a fraction: '" + nIndex +
            break;
        }
        else if (nIndex <= 0) {
            env.throwError("Array size for array '" + aryName + "' must be positive and non-zero: '" +
            break;
        }

        env.setValue(aryName.toString(), (int) nIndex, var_value1.retLast.toString());
    }
    else {
        env.setValue(var_name.toString(), var_value1.retLast.toString());
    }

    r = var_value1.retLast;
    current_node.retLast = r;
}
| var:ID
{
    int t = env.varCheck(var.toString());

    if (t != env.ERROR) {
        r = new SpinnerRet(env.getDoubleValue(var.toString()));
        current_node.retLast = r;
    }
}
| #(ARRAY array_name:. ary_index:.) {

    ary_index.exprHelper(this, current_node);
    int nArrayType = env.varCheck(array_name.toString());

    if (nArrayType == nARRAY_DOUBLE_TYPE ||
        nArrayType == nARRAY_BOOLEAN_TYPE)
    {
        r = new SpinnerRet(env.getDoubleValue(array_name.toString(),
                            (int) ary_index.retLast.doubleValue()));
    }
    else {
        assert(false);
    }

    current_node.retLast = r;
}
| #(PARAMS params_expr:. ) {

    while (params_expr != null)
    {
        params_expr.exprHelper(this, current_node);
        params_expr = (SpinnerAST) params_expr.getNextSibling();
    }

    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| #(PARALLEL .)
{
    InterpreterHelper.invokeParallel(env, sim, this, #PARALLEL);
    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| NUMBER {
    r = new SpinnerRet(#NUMBER.getText());
    current_node.retLast = r;
}
| "double"
| "boolean"
| "void"
| "true" {
    r = SpinnerRet.retTRUE;
    current_node.retLast = r;
}
| "false" {
    r = SpinnerRet.retFALSE;
    current_node.retLast = r;
```

```
}
| "null" {
    r = SpinnerRet.retVOID;
    current_node.retLast = r;
}
| "all"
| "master"
| LBRACE { assert(false); current_node.enterScope(env); }
| RBRACE { assert(false); current_node.leaveScope(env); }
;
```

D.4.2 InterpreterHelper.java

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/InterpreterHelper.java,v 1.2 2003/12/17 14:06:00 me13
//
//    File name: InterpreterHelper.java
//   Created By: Marc Eaddy
//   Created On: December 1, 2003, 11:18 PM
//
//      Purpose:
//
//  Description:
//
//////////////////////////////////////////////////////////////////////
/*
 * $Log: InterpreterHelper.java,v $
 * Revision 1.2  2003/12/17 14:06:00  me133
 * o Now supports nested parallel statements and pausing and resuming statements
 *
 * Revision 1.1  2003/12/02 05:08:42  me133
 * o Created
 */

package spinner.compiler;

import antlr.collections.AST;
import antlr.RecognitionException;

/**
 * @author  Marc Eaddy
 */
public class InterpreterHelper {

    public static SpinnerRet invokeFunction( Environment env,
Simulator sim,
SpinnerInterpreter interpreter,
String func_name,
SpinnerAST current_node,
SpinnerAST params)
        throws RecognitionException, PausedException
    {
        SpinnerFunction func = env.getFunction(func_name, null, false);
        if (func == null) return null; // SemanticAnalyzer already verified this
```

```
String[] paramNames = func.getParamNames();

try
{
if (paramNames != null) {

assert(params != null || paramNames.length == 0);

params.enterScope(env);

SpinnerAST param = (SpinnerAST) params.getFirstChild();

for(int i = 0; paramNames != null && i < paramNames.length; ++i) {
assert(param != null); // SemanticAnalyzer already verified this
param.exprHelper(interpreter, params);
env.addVariable(paramNames[i], Environment.nDOUBLE_TYPE, param.retLast.toString());
param = (SpinnerAST) param.getNextSibling();
 }
}

return func.invoke(interpreter, sim, current_node, params);
}
finally {
if (paramNames != null)
params.leaveScope(env);
}
    }

    public static void invokeParallel( Environment env,
Simulator sim,
SpinnerInterpreter interpreter,
SpinnerAST parallelNode)
        throws RecognitionException, PausedException
    {
env.enterParallel();

try {

boolean bAtLeastOneWaitingBranch = false;

do
{
bAtLeastOneWaitingBranch = false;

SpinnerAST nextBranchNode = (SpinnerAST) parallelNode.getFirstChild();
```

```
int i = 0;
while (nextBranchNode != null)
{
if (env.bDebug)
{
System.out.print("[" + (int) sim.getTime() + "] ");

for(int z = 0; z < env.getParallelNestingLevel()-1; ++z)
System.out.print("\t");

System.out.println("BRANCH " +
env.getParallelNestingLevel() +
"-" + i + ": " +
nextBranchNode.toStringList());
}

try {
nextBranchNode.enterScope(env);
if (nextBranchNode.retLast == null) {
nextBranchNode.retLast = interpreter.expr(nextBranchNode);
}
}
catch (PausedException ex) {

nextBranchNode.setNextNode(ex.getNext());
bAtLeastOneWaitingBranch = true;
}
finally {
nextBranchNode.leaveScope(env);
}

// Go to the next branch
nextBranchNode = (SpinnerAST) nextBranchNode.getNextSibling();
++i; // Just for diagnostics

} // while (parallelNode.nodeNext != null)

if (bAtLeastOneWaitingBranch)
{
if (env.isInTopParallelBranch())
{
sim.play(1);
}
else
{
```

```
throw new PausedException(parallelNode);
}
}
} while (bAtLeastOneWaitingBranch);


}
finally {
env.leaveParallel();
}
}

    public static SpinnerRet declareVariable( Environment env,
SpinnerInterpreter interpreter,
SpinnerAST current_node,
AST var_type,
AST decl_body)
        throws RecognitionException, PausedException
    {
SpinnerRet retVal = null;

AST name = null;

if (decl_body.getNumberOfChildren() == 0)
{
name = (AST) decl_body;
}
else
{
name = (AST) decl_body.getFirstChild();
}

assert(name != null);

String strInitialValue = null;
String strArraySize = null;
int nType = SpinnerInterpreter.nINIT_TYPE;

if (var_type.getText().equals("double"))
{
if (decl_body.getType() == SpinnerInterpreter.ARRAY)
nType = SpinnerInterpreter.nARRAY_DOUBLE_TYPE;
else
nType = SpinnerInterpreter.nDOUBLE_TYPE;
}
else if (var_type.getText().equals("boolean"))
```

```
{
if (decl_body.getType() == SpinnerInterpreter.ARRAY)
nType = SpinnerInterpreter.nARRAY_BOOLEAN_TYPE;
else
nType = SpinnerInterpreter.nBOOLEAN_TYPE;
}
else {
env.throwError(402, name.toString());
return null;
}

if (decl_body.getType() == SpinnerInterpreter.ARRAY)
{
SpinnerAST sizeNode = (SpinnerAST) name.getNextSibling();
assert(sizeNode != null);
sizeNode.exprHelper(interpreter, current_node);
double size = sizeNode.retLast.doubleValue();

if (size != (int) size) {
env.throwError("Array size cannot be a fraction: '" + size + "'.");
return null;
}
else if (size <= 0) {
env.throwError("Array size must be positive and non-zero: '" + size + "'.");
return null;
}

strArraySize = Integer.toString((int)size);
}
else if (decl_body.getType() == SpinnerInterpreter.ASSIGN)
{
SpinnerAST valNode = (SpinnerAST) name.getNextSibling();
assert(valNode != null);

valNode.exprHelper(interpreter, current_node);

if (nType == SpinnerInterpreter.nDOUBLE_TYPE) {
strInitialValue = valNode.retLast.toString();
}
else if (nType == SpinnerInterpreter.nBOOLEAN_TYPE) {
strInitialValue = valNode.retLast.doubleValue() != 0 ? "true" : "false";
}
}

if (strArraySize == null)
env.addVariable(name.toString(), nType, strInitialValue);
```

```
else
env.addVariable(name.toString(), nType, strArraySize, strInitialValue);

return strInitialValue != null ? new SpinnerRet(strInitialValue) : SpinnerRet.retVOID;
}

    public static int safeCompareTo(Double lhs, Double rhs) {
        if (lhs == null && rhs == null)
            return 0;
        else if (lhs == null)
            return -1;
        else if (rhs == null)
            return 1;
        else
            return lhs.compareTo(rhs);
    }

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
Main.main(args);
}
}
```

D.4.3 OneLineFormatter.java

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/OneLineFormatter.java,v 1.1 2003/11/11 06:31:13 me133
//
//    File name: OneLineFormatter.java
//   Created By: Marc Eaddy
//   Created On: November 10, 2003, 9:03 PM
//
//      Purpose: Formats log messages to fit on one line
//
//  Description: Also includes filename, class, and line number
// for the line where the log method was called.
// This makes it easy to to determine which source
// code line generated the log message.
//
////////////////////////////////////////////////////////////////////////
/*
 * $Log: OneLineFormatter.java,v $
 * Revision 1.1  2003/11/11 06:31:13  me133
 * o Created
 *
 */

package spinner.compiler;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Formatter;
import java.util.logging.LogRecord;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Logger;
import java.util.logging.Level;

/**
 * @author  Marc Eaddy
 */
public class OneLineFormatter extends Formatter {

// Stack frame output formats
```

```
/** Ex: spinner.compiler.OneLineFormatter.test(OneLineFormatter.java:151) */
static final int nSTANDARD = 0;

/** Ex: OneLineFormatter.test(OneLineFormatter.java:151) */
static final int nSTANDARD_COMPACT = 1;

/** Ex: File: OneLineFormatter.java, Line: 160 */
static final int nILX = 2;

/** Ex: File: OneLineFormatter.java::test, Line: 164 */
static final int nILX_WITH_METHOD = 3;

/** Ex: OneLineFormatter.java::test:164 */
static final int nULTRA_COMPACT = 4;

int stackFrameFormat = nILX_WITH_METHOD;

/** Creates a new instance of OneLineFormatter */
public OneLineFormatter() {
}

public String format(LogRecord record) {
StringBuffer out = new StringBuffer();

SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd HH:mm:ss");

out.append(dateFormat.format(new Date(record.getMillis())));
out.append(" ");
out.append(record.getThreadID());
out.append(" - ");
out.append(record.getLevel());
out.append(" - ");
out.append(record.getLoggerName());
out.append(" - ");
out.append(record.getMessage());

appendFileClassAndLine(out);

out.append("\n");

return out.toString();
}

void appendFileClassAndLine(StringBuffer out)
{
StackTraceElement ste[] = (new Throwable()).getStackTrace();
```

```
int nCallingStackFrame = -1;

// Half to walk backwards because the first stack frame
// is the last item in the array.  We only want to catch
// the first Logger.log() method since it gets called
// multiple times when a message is logged and therefore
// appears multiple times in the stack.
for(int i = ste.length-1; i >= 0 ; --i)
{
if (ste[i].getClassName().endsWith(".Logger") &&
ste[i].getMethodName().equals("log"))
{
// The Logger.log() method *appears* to be
// called by the info, warning, and other
// logging methods.  This means it is two
// stack frames away from the source line where
// the message was logged.
nCallingStackFrame = i + 2;

break;
}
}

assert(nCallingStackFrame >= 0 && nCallingStackFrame < ste.length);

out.append(" [");

// Use the standard Java format for displaying the stack frame
if (stackFrameFormat == nSTANDARD)
{
out.append(ste[nCallingStackFrame]);
}
else if (stackFrameFormat == nSTANDARD_COMPACT)
{
String strClassName = ste[nCallingStackFrame].getClassName();
int posLastPeriod = strClassName.lastIndexOf('.');
if (posLastPeriod > -1)
{
strClassName = strClassName.substring(posLastPeriod+1);
}

out.append(strClassName);
out.append(".");
out.append(ste[nCallingStackFrame].getMethodName());
out.append("(");
out.append(ste[nCallingStackFrame].getFileName());
out.append(":");
out.append(ste[nCallingStackFrame].getLineNumber());
out.append(")");
}
else if (stackFrameFormat == nILX)
{
out.append("File: ");
out.append(ste[nCallingStackFrame].getFileName());
out.append(", Line: ");
out.append(ste[nCallingStackFrame].getLineNumber());
}
else if (stackFrameFormat == nILX_WITH_METHOD)
{
out.append("File: ");
out.append(ste[nCallingStackFrame].getFileName());
out.append("::");
out.append(ste[nCallingStackFrame].getMethodName());
out.append(", Line: ");
out.append(ste[nCallingStackFrame].getLineNumber());
}
else if (stackFrameFormat == nULTRA_COMPACT)
{
out.append(ste[nCallingStackFrame].getFileName());
out.append("::");
out.append(ste[nCallingStackFrame].getMethodName());
out.append(":");
out.append(ste[nCallingStackFrame].getLineNumber());
}
else
{
assert("Illegal value for stackFrameForamt." == null);
}

out.append("]");
}

public static void test() {
Logger.getLogger("Test").info("Test message 1");
Logger.getLogger("Test").warning("Test message 2");
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
```

```
Handler fileHandler = null;

try {
fileHandler = new FileHandler("Test.log");
fileHandler.setFormatter(new OneLineFormatter());
}
catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);
}

assert(fileHandler != null);

Logger logger = Logger.getLogger("Test");
if (logger == null)
{
System.err.println("Failed to obtain logger object. Exiting.");
System.exit(-1);
}

logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);

test();
}
}
```

D.4.4 SpinnerFunction.java

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SpinnerFunction.java,v 1.10 2003/12/17 14:53:14 ss202
//
//    File name: SpinnerFunction.java
//   Created By: Marc Eaddy
//   Created On: November 21, 2003, 10:15 PM
//
//      Purpose:
//
//  Description: See below
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: SpinnerFunction.java,v $
 * Revision 1.10  2003/12/17 14:53:14  ss2020
 * Added 'import' function definition.
 *
 * Revision 1.9  2003/12/17 14:06:00  me133
 * o Now supports nested parallel statements and pausing and resuming statements
 *
 * Revision 1.8  2003/12/01 06:10:17  me133
 * o Fixed some javadoc comment bugs
 *
 * Revision 1.7  2003/11/30 20:53:04  me133
 * o Fixed compile error
 * o Builtin functions don't have parameter names
 *
 * Revision 1.6  2003/11/30 20:09:24  ss2020
 * Added getSymbolTable() and setSymbolTable().
 *
 * Revision 1.5  2003/11/30 19:51:16  me133
 * o Now uses same constants from SpinnerWalker
 * o Now holds the parameter names
 * o Fixed bug where calling setSlide(all, ...) twice failed the second time
 * o Now specifies the correct types when registering builtin functions with Environment
 * o Added getParamNames() and getRetType()
 *
 * Revision 1.4  2003/11/24 05:18:42  me133
 * o Added some asserts.  User defined functions now work.
 *
 * Revision 1.3  2003/11/24 04:44:39  me133
 * o Working on fixing user-defined function calls.  Not done yet.
 *
 * Revision 1.2  2003/11/24 00:52:40  me133
 * o Added getParams()
 *
 * Revision 1.1  2003/11/23 09:56:35  me133
 * o Created
 */

package spinner.compiler;

import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.MismatchedTokenException;
import antlr.TreeParser;

/**
 * Handles builtin Spinner functions and user-defined
 * functions.
 * <P>
 * Provides an abstraction/layer that hides whether a function
 * is builtin or user-defined.  Both are identical from the
 * Spinner programmer's point of view.  However, a user-defined
 * function is invoked by evaluating the abstract syntax tree (AST)
 * of the function body whereas builtin's are evaluated by calling
 * the appropriate Simulator class method.
 * <P>
 * @author  me133
 * @see "Spinner Language Reference Manual (LRM)"
 */
public class SpinnerFunction {

public static final int nINIT_TYPE = SpinnerWalker.nINIT_TYPE;
public static final int nDOUBLE_TYPE = SpinnerWalker.nDOUBLE_TYPE;
public static final int nBOOLEAN_TYPE = SpinnerWalker.nBOOLEAN_TYPE;
public static final int nVOID_TYPE = SpinnerWalker.nVOID_TYPE;
public static final int nSTRING_TYPE = SpinnerWalker.nSTRING_TYPE;

Simulator sim = null;
SpinnerInterpreter interpreter = null;
int retType = nINIT_TYPE;
String funcName = null;
int[] paramTypes;
String[] paramNames;
SpinnerAST body = null;
SymbolTable symbolTable  = null;
```

```
/**
 * Creates a user-defined Spinner function.
 */
public SpinnerFunction( int retType,
String funcName,
int[] paramTypes,
String[] paramNames,
SpinnerAST body)
{
this.retType = retType;
this.funcName = funcName;
this.paramTypes = paramTypes;
this.paramNames = paramNames;
this.body = body;

assert(funcName != null);
assert(funcName.length() > 0);
}

/**
 * Creates a builtin Spinner function.
 */
public SpinnerFunction( int retType,
String funcName,
int[] paramTypes)
{
this.retType = retType;
this.funcName = funcName;
this.paramTypes = paramTypes;
this.paramNames = null; // This is a builtin function
this.body = null; // This is a builtin function

assert(funcName != null);
assert(funcName.length() > 0);
}

public void setSymbolTable(SymbolTable table) {
symbolTable = table;
}

public SymbolTable getSymbolTable() {
return symbolTable;
}

/* This funtion is just for test */
public SpinnerAST getFuncBody() {
```

```
return body;
}

public SpinnerRet invoke( SpinnerInterpreter interpreter,
Simulator sim,
SpinnerAST current_node,
SpinnerAST nodeParams)
throws MismatchedTokenException, RecognitionException, PausedException
{
// I know this is bad, please forgive me!
this.sim = sim;
this.interpreter = interpreter;

if (body != null)
{
return invokeUserDefined(current_node);
}
else
{
return new SpinnerRet(invokeBuiltin(current_node, nodeParams));
}
}

private Double invokeBuiltin(SpinnerAST current_node, SpinnerAST nodeParams)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeParams != null);

if (funcName == null) {
System.err.println("Function name is null.");
return null;
}
else if (funcName.length() == 0 ) {
System.err.println("Function name is empty.");
return null;
}
else if (nodeParams == null) {
System.err.println("Function node is null.");
return null;
}

SpinnerAST nodeFirstParam = (SpinnerAST) nodeParams.getFirstChild();
nodeParams = null;

String funcNameCopy = funcName;
```

```
// Convert "all" to xxxAll()
if (funcNameCopy.equals("setGain") ||
funcNameCopy.equals("setSpinMult") ||
funcNameCopy.equals("setSampleStart") ||
funcNameCopy.equals("setLength") ||
funcNameCopy.equals("seek") ||
funcNameCopy.equals("setSpeed") ||
funcNameCopy.equals("setSlide") ||
funcNameCopy.equals("setFile")) {

matchNonNull(nodeFirstParam);

if (nodeFirstParam.getText().equals("all"))
{
funcNameCopy = funcNameCopy + "All";
nodeFirstParam = (SpinnerAST) nodeFirstParam.getNextSibling();
}
}

// Convert "master" to xxxMaster()
if (funcNameCopy.equals("setGain") ||
funcNameCopy.equals("getGain")) {

matchNonNull(nodeFirstParam);

if (nodeFirstParam.getText().equals("master"))
{
funcNameCopy = funcNameCopy + "Master";
nodeFirstParam = (SpinnerAST) nodeFirstParam.getNextSibling();
}
}

if (funcNameCopy.equals("getDiscsSize")) {
return sim.getDiscsSize();
}
//else if (funcNameCopy.equals("end")) { // Can't be called externally
//}
else if (funcNameCopy.equals("getTime")) {
return new Double(sim.getTime());
}
else if (funcNameCopy.equals("getGainMaster")) {
return sim.getGainMaster();
}
else if ( funcNameCopy.equals("createDiscs") ||
funcNameCopy.equals("getSpeed") ||
funcNameCopy.equals("getPosition") ||
funcNameCopy.equals("wait") ||
funcNameCopy.equals("setGainMaster") ||
funcNameCopy.equals("setGainAll") ||
funcNameCopy.equals("getGain") ||
funcNameCopy.equals("setSpinMultAll") ||
funcNameCopy.equals("getSpinMult") ||
funcNameCopy.equals("setSampleStartAll") ||
funcNameCopy.equals("getSampleStart") ||
funcNameCopy.equals("setLengthAll") ||
funcNameCopy.equals("getLength") ||
funcNameCopy.equals("isFileSet") ||
funcNameCopy.equals("commentVar"))
{
// In some cases, the return value for the builtin is
// void.  In these cases we just return null.
return invokeBuiltin_Num_Double(current_node, funcNameCopy, nodeFirstParam);
}
else if ( funcNameCopy.equals("seek") ||
funcNameCopy.equals("setSpeed") ||
funcNameCopy.equals("commentVar3"))
{
invokeBuiltin_NumNumNum_Void(current_node, funcNameCopy, nodeFirstParam);
return null;
}
else if ( funcNameCopy.equals("seekAll") ||
funcNameCopy.equals("setSpeedAll") ||
funcNameCopy.equals("setGain") ||
funcNameCopy.equals("setSpinMult") ||
funcNameCopy.equals("setSampleStart") ||
funcNameCopy.equals("setLength") ||
funcNameCopy.equals("commentVar2"))
{
invokeBuiltin_NumNum_Void(current_node, funcNameCopy, nodeFirstParam);
return null;
}
else if ( funcNameCopy.equals("setSlide") ||
funcNameCopy.equals("commentNumTextNum")) {
invokeBuiltin_NumTextNum_Void(current_node, funcNameCopy, nodeFirstParam);
return null;
}
else if ( funcNameCopy.equals("setSlideAll") ||
funcNameCopy.equals("setReverb") ||
funcNameCopy.equals("setSub") ||
funcNameCopy.equals("commentTextNum")) {
invokeBuiltin_TextNum_Void(current_node, funcNameCopy, nodeFirstParam);
```

```
return null;
}
else if ( funcNameCopy.equals("getSlide") ||
funcNameCopy.equals("setFile") ||
funcNameCopy.equals("commentNumText")) {
// In some cases, the return value for the builtin is
// void.  In these cases we just return null.
return invokeBuiltin_NumText_Double(current_node, funcNameCopy, nodeFirstParam);
}
else if ( funcNameCopy.equals("getReverb") ||
funcNameCopy.equals("getSub") ||
funcNameCopy.equals("setFileAll") ||
funcNameCopy.equals("comment")) {
// In some cases, the return value for the builtin is
// void.  In these cases we just return null.
return invokeBuiltin_Text_Double(current_node, funcNameCopy, nodeFirstParam);
}
else {
System.err.println("Unknown builtin function. '" + funcNameCopy + "'.");
return null;
}
}

private SpinnerRet invokeUserDefined(SpinnerAST current_node)
throws RecognitionException, PausedException
{
assert(body != null);
assert(interpreter != null);
assert(current_node != null);
SpinnerRet ret = body.exprHelper(interpreter, current_node);
body.forgetReturnValue();
return ret;
}

// *******************************************************
// Helper functions for unwrapping arguments and calling
// Builtin functions
// *******************************************************

private Double invokeBuiltin_Num_Double(SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);
```

```
SpinnerAST nodeArg1 = nodeFirstParam;

nodeArg1.exprHelper(interpreter, current_node);

if (nodeArg1.retLast == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (funcNameToUse == null) {
assert(false); // Can't happen since caller should have filtered
System.err.println("Function name is null.");
return null;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can't happen since caller should have filtered
System.err.println("Function name is empty.");
return null;
}
else if (funcNameToUse.equals("createDiscs")) {
sim.createDiscs(nodeArg1.retLast.intValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("getSpeed")) {
return new Double(sim.getSpeed(nodeArg1.retLast.intValue()));
}
else if (funcNameToUse.equals("getPosition")) {
return new Double(sim.getPosition(nodeArg1.retLast.intValue()));
}
else if (funcNameToUse.equals("wait")) {
sim.play(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("setGainMaster")) {
sim.setGainMaster(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("setGainAll")) {
sim.setGainAll(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("getGain")) {
return sim.getGain(nodeArg1.retLast.intValue());
}
else if (funcNameToUse.equals("setSpinMultAll")) {
sim.setSpinMultAll(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("getSpinMult")) {
return sim.getSpinMult(nodeArg1.retLast.intValue());
}
else if (funcNameToUse.equals("setSampleStartAll")) {
sim.setSampleStartAll(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("getSampleStart")) {
return sim.getSampleStart(nodeArg1.retLast.intValue());
}
else if (funcNameToUse.equals("setLengthAll")) {
sim.setLengthAll(nodeArg1.retLast.doubleValue());
return null; // void function so just return null
}
else if (funcNameToUse.equals("getLength")) {
return sim.getLength(nodeArg1.retLast.intValue());
}
else if (funcNameToUse.equals("isFileSet")) {
return new Double(sim.isFileSet(nodeArg1.retLast.intValue()) ? 1.0 : 0.0);
}
else if (funcNameToUse.equals("commentVar")) {
sim.outputComment(nodeArg1.toString() + "=" + nodeArg1.retLast.toString());
return null;
}
else
{
assert(false); // Can't happen since caller should have filtered
System.err.println("Unknown builtin function. '" + funcNameToUse + "'.");
return null;
}
}

private Double invokeBuiltin_NumText_Double(SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
matchNonNull(nodeArg1);
SpinnerAST nodeArg2 = (SpinnerAST) nodeArg1.getNextSibling();
```

```
nodeArg1.exprHelper(interpreter, current_node);

if (nodeArg1.retLast == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (nodeArg2 == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (funcNameToUse == null) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is null.");
return null;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is empty.");
return null;
}
else if (funcNameToUse.equals("getSlide")) {
return sim.getSlide( nodeArg1.retLast.intValue(), nodeArg2.getText() );
}
else if (funcNameToUse.equals("setFile")) {
sim.setFile( nodeArg1.retLast.intValue(), nodeArg2.getText() );
return null; // void function so just return null
}
else if (funcNameToUse.equals("commentNumText")) {
sim.outputComment( new Double(nodeArg1.retLast.doubleValue()), nodeArg2.getText());
return null;
}
else
{
assert(false); // Can’t happen since caller should have filtered
System.err.println("Unknown builtin function. ’" + funcNameToUse + "’.");
return null;
}
}

private Double invokeBuiltin_Text_Double(SpinnerAST current_node, String funcNameToUse, SpinnerAST nodeF
throws MismatchedTokenException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
```

```
if (nodeArg1 == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (funcNameToUse == null) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is null.");
return null;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is empty.");
return null;
}
else if (funcNameToUse.equals("getReverb")) {
return sim.getReverb(nodeArg1.getText());
}
else if (funcNameToUse.equals("getSub")) {
return sim.getSub(nodeArg1.getText());
}
else if (funcNameToUse.equals("setFileAll")) {
sim.setFileAll(nodeArg1.getText());
return null; // void function so just return null
}
else if (funcNameToUse.equals("comment")) {
sim.outputComment(nodeArg1.getText());
return null; // void function so just return null
}
else
{
assert(false); // Can’t happen since caller should have filtered
System.err.println("Unknown builtin function. ’" + funcNameToUse + "’.");
return null;
}
}

private void invokeBuiltin_NumNum_Void( SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
```

```
matchNonNull(nodeArg1);
SpinnerAST nodeArg2 = (SpinnerAST) nodeArg1.getNextSibling();

nodeArg1.exprHelper(interpreter, current_node);
nodeArg2.exprHelper(interpreter, current_node);

if (nodeArg1.retLast == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (nodeArg2.retLast == null) {
throw new IllegalArgumentException("nodeArg2.retLast cannot be null.");
}

if (funcNameToUse == null) {
System.err.println("Function name is null.");
return;
}
else if (funcNameToUse.length() == 0 ) {
System.err.println("Function name is empty.");
return;
}
else if (funcNameToUse.equals("seekAll")) {
sim.seekAll(nodeArg1.retLast.doubleValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setSpeedAll")) {
sim.setSpeedAll(nodeArg1.retLast.doubleValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setGain")) {
sim.setGain(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setSpinMult")) {
sim.setSpinMult(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setSampleStart")) {
sim.setSampleStart(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setLength")) {
sim.setLength(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("commentVar2")) {
sim.outputComment( nodeArg1.toString() + "=" + nodeArg1.retLast.toString() + ", " +
nodeArg2.toString() + "=" + nodeArg2.retLast.toString());
}
else
{
```

```
assert(false); // Can’t happen since caller should have filtered
System.err.println("Unknown builtin function. ’" + funcNameToUse + "’.");
}
}

private void invokeBuiltin_NumNumNum_Void( SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
matchNonNull(nodeArg1);
SpinnerAST nodeArg2 = (SpinnerAST) nodeArg1.getNextSibling();
matchNonNull(nodeArg2);
SpinnerAST nodeArg3 = (SpinnerAST) nodeArg2.getNextSibling();

nodeArg1.exprHelper(interpreter, current_node);
nodeArg2.exprHelper(interpreter, current_node);
nodeArg3.exprHelper(interpreter, current_node);

if (nodeArg1.retLast == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (nodeArg2.retLast == null) {
throw new IllegalArgumentException("nodeArg2.retLast cannot be null.");
}

if (nodeArg3.retLast == null) {
throw new IllegalArgumentException("nodeArg3.retLast cannot be null.");
}

if (funcNameToUse == null) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is null.");
return;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is empty.");
return;
}
else if (funcNameToUse.equals("seek")) {
```

```
sim.seek(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue(), nodeArg3.retLast.doubleValue());
}
else if (funcNameToUse.equals("setSpeed")) {
sim.setSpeed(nodeArg1.retLast.intValue(), nodeArg2.retLast.doubleValue(), nodeArg3.retLast.doubleValue(
}
else if (funcNameToUse.equals("commentVar3")) {
sim.outputComment( nodeArg1.toString() + "=" + nodeArg1.retLast.toString() + ", " +
nodeArg2.toString() + "=" + nodeArg2.retLast.toString() + ", " +
nodeArg3.toString() + "=" + nodeArg3.retLast.toString());
}
else
{
assert(false); // Can't happen since caller should have filtered
System.err.println("Unknown builtin function. '" + funcNameToUse + "'.");
}
}

private void invokeBuiltin_NumTextNum_Void( SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
matchNonNull(nodeArg1);
SpinnerAST nodeArg2 = (SpinnerAST) nodeArg1.getNextSibling();
matchNonNull(nodeArg2);
SpinnerAST nodeArg3 = (SpinnerAST) nodeArg2.getNextSibling();

nodeArg1.exprHelper(interpreter, current_node);
nodeArg3.exprHelper(interpreter, current_node);

if (nodeArg1.retLast == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (nodeArg2 == null) {
throw new IllegalArgumentException("nodeArg2.retLast cannot be null.");
}

if (nodeArg3.retLast == null) {
throw new IllegalArgumentException("nodeArg3.retLast cannot be null.");
}
```

```
if (funcNameToUse == null) {
assert(false); // Can't happen since caller should have filtered
System.err.println("Function name is null.");
return;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can't happen since caller should have filtered
System.err.println("Function name is empty.");
return;
}
else if (funcNameToUse.equals("setSlide")) {
sim.setSlide(nodeArg1.retLast.intValue(), nodeArg2.getText(), nodeArg3.retLast.doubleValue());
}
else if (funcNameToUse.equals("commentNumTextNum")) {
sim.outputComment( new Double(nodeArg1.retLast.doubleValue()),
nodeArg2.getText(),
new Double(nodeArg3.retLast.doubleValue()));
}
else
{
assert(false); // Can't happen since caller should have filtered
System.err.println("Unknown builtin function. '" + funcNameToUse + "'.");
}
}

private void invokeBuiltin_TextNum_Void(SpinnerAST current_node,
String funcNameToUse,
SpinnerAST nodeFirstParam)
throws MismatchedTokenException, RecognitionException, PausedException
{
assert(sim != null);
assert(nodeFirstParam != null);

SpinnerAST nodeArg1 = nodeFirstParam;
matchNonNull(nodeArg1);
SpinnerAST nodeArg2 = (SpinnerAST) nodeArg1.getNextSibling();

nodeArg2.exprHelper(interpreter, current_node);

if (nodeArg1 == null) {
throw new IllegalArgumentException("nodeArg1.retLast cannot be null.");
}

if (nodeArg2.retLast == null) {
throw new IllegalArgumentException("nodeArg2.retLast cannot be null.");
}
```

```
if (funcNameToUse == null) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is null.");
return;
}
else if (funcNameToUse.length() == 0 ) {
assert(false); // Can’t happen since caller should have filtered
System.err.println("Function name is empty.");
return;
}
else if (funcNameToUse.equals("setSlideAll")) {
sim.setSlideAll(nodeArg1.getText(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setReverb")) {
sim.setReverb(nodeArg1.getText(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("setSub")) {
sim.setSub(nodeArg1.getText(), nodeArg2.retLast.doubleValue());
}
else if (funcNameToUse.equals("commentTextNum")) {
sim.outputComment( nodeArg1.getText(),
new Double(nodeArg2.retLast.doubleValue()));
}
else
{
assert(false); // Can’t happen since caller should have filtered
System.err.println("Unknown builtin function. ’" + funcNameToUse + "’.");
}
}

// HELPER FUNCTIONS

private void matchNonNull(AST t) throws MismatchedTokenException {
        if (t == null || t == TreeParser.ASTNULL) {
            throw new MismatchedTokenException();
        }
    }

static void addBuiltins(Environment env) {
assert(env != null);

env.addBuiltinFunction(nDOUBLE_TYPE,"getDiscsSize", getParamTypes());
env.addBuiltinFunction(nDOUBLE_TYPE,"getTime", getParamTypes());
env.addBuiltinFunction(nDOUBLE_TYPE,"getGainMaster", getParamTypes());
env.addBuiltinFunction(nVOID_TYPE,"createDiscs", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getSpeed", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getPosition", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"wait", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setGainMaster", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setGainAll", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getGain", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSpinMultAll", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getSpinMult", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSampleStartAll", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getSampleStart", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setLengthAll", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getLength", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nBOOLEAN_TYPE,"isFileSet", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"seek", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSpeed", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"seekAll", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSpeedAll", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setGain", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSpinMult", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSampleStart", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setLength", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSlide", getParamTypes(nDOUBLE_TYPE, nSTRING_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSlideAll", getParamTypes(nSTRING_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setReverb", getParamTypes(nSTRING_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setSub", getParamTypes(nSTRING_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getSlide", getParamTypes(nDOUBLE_TYPE, nSTRING_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setFile", getParamTypes(nDOUBLE_TYPE, nSTRING_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getReverb", getParamTypes(nSTRING_TYPE));
env.addBuiltinFunction(nDOUBLE_TYPE,"getSub", getParamTypes(nSTRING_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"setFileAll", getParamTypes(nSTRING_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"import", getParamTypes(nSTRING_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"comment", getParamTypes(nSTRING_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"commentVar", getParamTypes(nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"commentVar2", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"commentVar3", getParamTypes(nDOUBLE_TYPE, nDOUBLE_TYPE, nDOUBLE_TYPE
env.addBuiltinFunction(nVOID_TYPE,"commentNumTextNum", getParamTypes(nDOUBLE_TYPE, nSTRING_TYPE, nDOUBL
env.addBuiltinFunction(nVOID_TYPE,"commentTextNum", getParamTypes(nSTRING_TYPE, nDOUBLE_TYPE));
env.addBuiltinFunction(nVOID_TYPE,"commentNumText", getParamTypes(nDOUBLE_TYPE, nSTRING_TYPE));

}

public int[] getParams() {
return paramTypes;
}

public String[] getParamNames() {
```

```
return paramNames;
}

public int getRetType() {
return retType;
}

static int[] getParamTypes() {
return null;
}

static int[] getParamTypes(int nArgType1) {
return new int[] { nArgType1 };
}

static int[] getParamTypes(int nArgType1, int nArgType2) {
return new int[] { nArgType1, nArgType2 };
}

static int[] getParamTypes(int nArgType1, int nArgType2, int nArgType3) {
return new int[] { nArgType1, nArgType2, nArgType3 };
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
Main.main(args);
}
}
```

D.4.5 PausedException.java

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/PausedException.java,v 1.1 2003/12/17 02:16:29 me133
//
//    File name: PausedException.java
//   Created By: Marc Eaddy
//   Created On: December 8, 2003, 6:14 PM
//
//      Purpose:
//
//  Description:
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: PausedException.java,v $
 * Revision 1.1  2003/12/17 02:16:29  me133
 * o Created
 *
 */

package spinner.compiler;

public class PausedException extends Exception {
SpinnerAST next;

public PausedException(SpinnerAST next) {
this.next = next;
}

public SpinnerAST getNext() {
return next;
}
}
```

D.5 Simulator

D.5.1 Simulator.java

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Simulator.java,v 1.20 2003/12/17 15:54:11 me133 Exp S
//
//    File name: Simulator.java
//   Created By: Marc Eaddy
//   Created On: November 10, 2003, 1:40 AM
//
//      Purpose: Implements (almost) all SPINNER functions
//
//  Description: Implements all the SPINNER functions that query or manipulate
// the state of the simulation, ie the state of the discs.
//
//////////////////////////////////////////////////////////////////////
/*
 * $Log: Simulator.java,v $
 * Revision 1.20  2003/12/17 15:54:11  me133
 * o Added description comment
 *
 * Revision 1.19  2003/12/17 14:03:14  me133
 * o Now if debug is true we flush the spin script output after every line intead of buffering it
 *
 * Revision 1.18  2003/12/17 05:26:28  me133
 * o Reformatted back to 4-space tabs
 * o Added outputComment() methods.  You can now output comments from Spinner programs!
 *
 * Revision 1.17  2003/12/17 02:54:37  me133
 * o Trivial change
 *
 * Revision 1.16  2003/12/17 00:47:56  me133
 * o Fixed package name (should be spinner.compiler)
 *
 * Revision 1.15  2003/12/17 00:10:53  wmb2013
 * ran formatter
 *
 * Revision 1.14  2003/12/08 01:18:05  wmb2013
 * reset package declaration to spinner.compiler
 *
 * Revision 1.13  2003/12/08 00:57:54  wmb2013
 * fixes to setSpeedAll, setSeekAll to for coordination with SPIN
```

```
*
* Revision 1.12  2003/12/06 22:22:20  wmb2013
* modified seek() seekAll() setSpeed() and setSpeedAll() to force SPIN not to add speed and seek
*
* Revision 1.11  2003/12/03 01:27:49  me133
* o Fixed bug I introduced where I forgot to put the dashes for the wait command
*
* Revision 1.10  2003/12/01 22:30:30  me133
* o Now ends comments with semicolon
*
* Revision 1.9  2003/12/01 07:46:11  me133
* o MAXscript had a problem with using getTime() as a prefix.  Now uses "1," as the prefix and outputs
*
* Revision 1.8  2003/12/01 06:11:13  me133
* o Fixed some javadoc comment bugs
*
* Revision 1.7  2003/12/01 05:57:11  me133
* o Now aggregates/buffers wait commands
* o Fixed some javadoc comment bugs
* o Added some comments
*
* Revision 1.6  2003/11/30 19:52:16  me133
* o Fixed bug where passing "all" or "master" threw an exception
*
* Revision 1.5  2003/11/23 09:48:34  me133
* o Interpreter now passes index array's as int not double
* o Added more checks and error handling
*
* Revision 1.4  2003/11/17 01:40:58  me133
* o Fixed bug where we were returning "reverb" instead of "sub"
*
* Revision 1.3  2003/11/17 01:38:00  me133
* o Now returns null as per the LRM
* o main() now tries to create cue-01
* o Disc array indexing now starts at 1
* o Now implements "master" disc
* o No longer outputs log messages intermixed with SPINscript
* o Added "--end--" command
* o Now outputs the current simulation time instead of "1,"
*
* Revision 1.2  2003/11/11 06:32:48  me133
* o Log section shouldn't use javadoc style comments
* o Added comments
* o Organized methods better
* o Now logs all method calls
* o Now outputs SpinScript when appropriate
```

```
 * o Added a little test code inside main()
 *
 * Revision 1.1  2003/11/10 23:28:51  me133
 * o Created
 */

package spinner.compiler;

import java.io.PrintStream;
import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Logger;
import java.util.logging.Level;

/**
 * Implements all the SPINNER functions that query or manipulate
 * the state of the simulation, ie the state of the discs.
 * <P>
 * After initializing the simulation, for example, the speed of
 * each disc, you can "play" the simulation for a specific amount
 * of time.  Playing the simulation basically means allowing time
 * to pass.  Since each disc has an associated speed, after time
 * has passed the discs will have rotated to different positions.
 * The simulation keeps tracks of things like the speeds and
 * positions of the discs so you can query for their values at
 * any time.
 * <P>
 * An important side-effect of calling methods on this class is
 * that the SPINscript equivalent command syntax will be output
 * to standard out.  Sometimes there is a one-to-one relationship.
 * For example, the SPINNER line
 * <PRE>
 *  setGainAll(0.25);
 * </PRE>
 * will cause the Interpreter to call the setGain() method on the
 * Simulator class.  The method will then output the following
 * SPINscript:
 * <PRE>
 * 1, all-gain 0.25;
 * </PRE>
 * In addition, for diagnostic purposes, all method calls and their
 * arguments are logged to Simulator.log.
 * <P>
 * The following Spinner built-in functions are not implemented by
 * Simulator because these should be implemented by the Interpreter:
```

```
 * <PRE>
 * waitUntil
 * import
 * getSize
 * </PRE>
 * @author  Marc Eaddy
 * @see "Spinner Language Reference Manual (LRM)"
 */
public class Simulator {

// ********************************************************
// Properties
// ********************************************************

private Disc[] discs = null;
private Disc discMaster = null;

private Double sendReverb = null;
private Double returnReverb = null;
private Double sendSub = null;
private Double returnSub = null;
private double currentTime = 0.0;

private PrintStream outSPINscript = null;
private Logger logger = null;

/**
 * Used when aggregating wait commands.
 * @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args)
 */
private double accumulatedWaitTime = 0.0;
private boolean bOutputFirstTimeComment = false;

private boolean bDisableOutputBuffering = false;

// ********************************************************
// CTOR
// ********************************************************

/** Creates a new instance of Simulator */
public Simulator(PrintStream outSPINscript, String logFileName) {
assert(outSPINscript != null);
assert(logFileName != null);
assert(logFileName.length() != 0);

this.outSPINscript = outSPINscript;
```

```
Handler fileHandler = null;

try {
fileHandler = new FileHandler(logFileName);
fileHandler.setFormatter(new OneLineFormatter());
} catch (IOException ex) {
ex.printStackTrace();
System.exit(-1);
}

assert(fileHandler != null);

this.logger = Logger.getLogger("Simulator");
if (logger == null) {
System.err.println("Failed to obtain logger object. Exiting.");
System.exit(-1);
}

logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);

// Modify the parent handlers so they also use the one
// line formatter.  These handlers output to the console.
Logger parent = logger.getParent();
Handler[] handlers = parent.getHandlers();
for (int i = 0; i < handlers.length; ++i) {
handlers[i].setFormatter(new OneLineFormatter());
}

// Set this to false to prevent the log messages from
// appearing on standard output
logger.setUseParentHandlers(false);
}

// *********************************************************
// Disc Array Functions
// *********************************************************

/**
 * Corresponds to the createDiscs Spinner built-in function.
 */
void createDiscs(int total) {

assert(logger != null);
```

```
// Log this method call
logger.info("createDiscs(" + total + ");");

assert(outSPINscript != null);

if (discs != null) {
System.err.println("ERROR: You can only call createDiscs once.");
return;
}

discMaster = new Disc(0, outSPINscript, logger);

discs = new Disc[total];

for (int i = 0; i < total; ++i) {
discs[i] = new Disc(i + 1, outSPINscript, logger);
}
}

/**
 * Corresponds to the getDiscsSize Spinner built-in function.
 */
Double getDiscsSize() {

assert(logger != null);

// Log this method call
logger.info("getDiscsSize();");

return discs == null ? null : new Double(discs.length);
}

// ********************************************************
// Disc Motion Functions
// ********************************************************

/**
 * Corresponds to the seek(discNum, ...) Spinner built-in function.
 */
void seek(int discNum, double seekPosition, double time) {

assert(logger != null);

// Log this method call
logger.info("seek(" + discNum + ", " + seekPosition + ", " + time + ");");
```

```
int discIndex = convertDiscNumToIndex(discNum);

//this forces SPIN to act like our simulator = prevents additive speeds/seeks
//if ever want to change SPINs behavior = change this
if (!discs[discIndex].isSeeking) {
outputSpinScriptDisc("speed", discNum, 0, 0);
discs[discIndex].setSpeed(0, 0);
}

Object[] args =
new Object[] { new Double(seekPosition), new Integer((int) time)};
outputSpinScriptDisc("line", discNum, args);

discs[discIndex].seek(seekPosition, time);
}

/**
 * Corresponds to the seek(all, ...) Spinner built-in function.
 */
void seekAll(double seekPosition, double time) {

assert(logger != null);

// Log this method call
logger.info("seekAll(" + seekPosition + ", " + time + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

Object[] args =
new Object[] { new Double(seekPosition), new Integer((int) time)};

for(int i = 0; i < discs.length; ++i) {
//this forces SPIN to act like our simulator = prevents additive speeds/seeks
//if ever want to change SPINs behavior = change this
if (!discs[i].isSeeking) {
discs[i].setSpeed(0, 0);
outputSpinScriptDisc("speed", i + 1, 0, 0); //also kill this
discs[i].seek(seekPosition, time);
}
}

outputSpinScript("line", "all", args);
}
```

```
/**
 * Corresponds to the setSoeed(discNum, ...) Spinner built-in function.
 */
void setSpeed(int discNum, double targetSpeed, double rampTime) {

assert(logger != null);

// Log this method call
logger.info(
"setSpeed(" + discNum + ", " + targetSpeed + ", " + rampTime + ");");

int discIndex = convertDiscNumToIndex(discNum);

//this forces SPIN to act like our simulator = prevents additive speeds/seeks
//if ever want to change SPINs behavior = change this
if (discs[discIndex].isSeeking) {
Object[] args = new Object[] {
new Double(discs[discIndex].getPosition()),
new Integer(0) };

outputSpinScriptDisc("line", discNum, args);
discs[discIndex].seek(discs[discIndex].getPosition(), 0);
}

//now output speed
outputSpinScriptDisc("speed", discNum, targetSpeed, rampTime);
discs[discIndex].setSpeed(targetSpeed, rampTime);
}

/**
 * Corresponds to the setSpeed(all, ...) Spinner built-in function.
 */
void setSpeedAll(double targetSpeed, double rampTime) {

assert(logger != null);

// Log this method call
logger.info("setSpeedAll(" + targetSpeed + ", " + rampTime + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

for(int i = 0; i < discs.length; ++i) {
//this forces SPIN to act like our simulator = prevents additive speeds/seeks
//if ever want to change SPINs behavior = change this
```

```
if (discs[i].isSeeking) {
Object[] args = new Object[] {
new Double(discs[i].getPosition()),
new Integer(0) };
outputSpinScriptDisc("line", i + 1, args);
discs[i].seek(discs[i].getPosition(), 0);
}

discs[i].setSpeed(targetSpeed, (int) rampTime);
}

outputSpinScript("speed", "all", targetSpeed, (int) rampTime);
}

/**
 * Corresponds to the getSpeed Spinner built-in function.
 */
double getSpeed(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getSpeed(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getSpeed();
}

/**
 * Corresponds to the setSlide(discNum, ...) Spinner built-in function.
 */
void setSlide(int discNum, String dir, double slide) {

assert(logger != null);

// Log this method call
logger.info("setSlide(" + discNum + ", " + dir + ", " + slide + ");");

int discIndex = convertDiscNumToIndex(discNum);

if (dir == null) {
throw new NullPointerException("Slide direction cannot be null.  Must be \"up\" or \"down\".");
}
else if (dir.equals("up")) {
outputSpinScriptDisc("slideup", discNum, slide);
```

```
discs[discIndex].setSlideUp(slide);
}
else if (dir.equals("down")) {
outputSpinScriptDisc("slidedown", discNum, slide);
discs[discIndex].setSlideDown(slide);
}
else {
throw new IllegalArgumentException("Illegal slide direction '"
+ dir
+ "'.  Must be \"up\" or \"down\".");
}
}

/**
 * Corresponds to the setSlide(all, ...) Spinner built-in function.
 */
void setSlideAll(String dir, double slide) {

assert(logger != null);

// Log this method call
logger.info("setSlideAll(" + dir + ", " + slide + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}
else if (dir == null) {
throw new NullPointerException("Slide direction cannot be null.  Must be \"up\" or \"down\".");
}
else if (dir.equals("up")) {
outputSpinScript("slideup", "all", slide);

for (int i = 0; i < discs.length; ++i) {
discs[i].setSlideUp(slide);
}
}
else if (dir.equals("down")) {
outputSpinScript("slidedown", "all", slide);

for (int i = 0; i < discs.length; ++i) {
discs[i].setSlideDown(slide);
}
}
else {
throw new IllegalArgumentException("Illegal slide direction: '"
+ dir
+ "'.  Must be \"up\" or \"down\".");
}
}

/**
 * Corresponds to the getSlide Spinner built-in function.
 */
Double getSlide(int discNum, String dir) {

assert(logger != null);

// Log this method call
logger.info("getSlide(" + discNum + ", " + dir + ");");

int discIndex = convertDiscNumToIndex(discNum);

if (dir == null) {
throw new NullPointerException("Slide direction cannot be null.  Must be \"up\" or \"down\".");
}
else if (dir.equals("up")) {
return discs[discIndex].getSlideUp();
}
else if (dir.equals("down")) {
return discs[discIndex].getSlideDown();
}
else {
throw new IllegalArgumentException("Illegal slide direction: '"
+ dir
+ "'.  Must be \"up\" or \"down\".");
}
}

/**
 * Corresponds to the getPosition Spinner built-in function.
 */
double getPosition(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getPosition(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getPosition();
}
```

```
// ********************************************************
// Simulation Functions
// ********************************************************

/**
 * Corresponds to the wait Spinner built-in function.
 */
void play(double time) {

assert(logger != null);

// Log this method call
logger.info("wait(" + time + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

assert(time >= 0);

for (int i = 0; i < discs.length; ++i) {
discs[i].play(time);
}

currentTime += time;

// Make sure you do this *after* updating currentTime!
Object[] args = new Object[] { new Integer((int) time)};
outputSpinScript("--wait--", null, args);
}

/**
 * Corresponds to the end Spinner built-in function.
 */
void end(double time) {

assert(logger != null);

// Log this method call
logger.info("end(" + time + ");");
Object[] args = new Object[] { new Integer((int) time)};
outputSpinScript("--end--", null, args);

assert(time >= 0);
```

```
currentTime += time;
}

/**
 * Corresponds to the getTime Spinner built-in function.
 */
double getTime() {

assert(logger != null);

// Log this method call
logger.info("getTime();");

return currentTime;
}

// ********************************************************
// Sound/Sample Functions (no affect on motion)
// ********************************************************

/**
 * Corresponds to the setGain(discNum, ...) Spinner built-in function.
 */
void setGain(int discNum, double gain) {

assert(logger != null);

// Log this method call
logger.info("setGain(" + discNum + ", " + gain + ");");

int discIndex = convertDiscNumToIndex(discNum);

outputSpinScriptDisc("gain", discNum, gain);

discs[discIndex].setGain(gain);
}

/**
 * Corresponds to the setGain(master, ...) Spinner built-in function.
 */
void setGainMaster(double gain) {

assert(logger != null);

// Log this method call
logger.info("setGainMaster(" + gain + ");");
```

```
outputSpinScript("gain", "master", gain);

discMaster.setGain(gain);
}

/**
 * Corresponds to the setGain(all, ...) Spinner built-in function.
 */
void setGainAll(double gain) {

assert(logger != null);

// Log this method call
logger.info("setGainAll(" + gain + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

outputSpinScript("gain", "all", gain);

for (int i = 0; i < discs.length; ++i) {
discs[i].setGain(gain);
}
}

/**
 * Corresponds to the getGain Spinner built-in function.
 */
Double getGain(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getGain(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getGain();
}

/**
 * Corresponds to the getGainMaster Spinner built-in function.
 */
Double getGainMaster() {
```

```
assert(logger != null);

// Log this method call
logger.info("getGainMaster();");

return discMaster.getGain();
}

/**
 * Corresponds to the setReverb Spinner built-in function.
 */
void setReverb(String sendOrReturn, double val) {

assert(logger != null);

// Log this method call
logger.info("setReverb(" + sendOrReturn + ", " + val + ");");

if (sendOrReturn == null) {
throw new IllegalArgumentException();
}
else if (sendOrReturn.equals("send")) {
sendReverb = new Double(val);
outputSpinScript("reverb-send", null, val);
}
else if (sendOrReturn.equals("return")) {
returnReverb = new Double(val);
outputSpinScript("reverb-return", null, val);
}
else {
throw new IllegalArgumentException("Illegal reverb argument #1: '"
+ sendOrReturn
+ "'.  Must be \"send\" or \"return\".");
}
}

/**
 * Corresponds to the getReverb Spinner built-in function.
 */
Double getReverb(String sendOrReturn) {

assert(logger != null);

// Log this method call
logger.info("getReverb(" + sendOrReturn + ");");
```

```
if (sendOrReturn == null) {
throw new IllegalArgumentException();
}
else if (sendOrReturn.equals("send")) {
return sendReverb;
}
else if (sendOrReturn.equals("return")) {
return returnReverb;
}
else {
throw new IllegalArgumentException("Illegal reverb argument #1: '"
+ sendOrReturn
+ "'.  Must be \"send\" or \"return\".");
}
}

/**
 * Corresponds to the setSub Spinner built-in function.
 */
void setSub(String sendOrReturn, double val) {

assert(logger != null);

// Log this method call
logger.info("setSub(" + sendOrReturn + ", " + val + ");");

if (sendOrReturn == null) {
throw new IllegalArgumentException();
}
else if (sendOrReturn.equals("send")) {
sendSub = new Double(val);
outputSpinScript("sub-send", null, val);
}
else if (sendOrReturn.equals("return")) {
returnSub = new Double(val);
outputSpinScript("sub-return", null, val);
}
else {
throw new IllegalArgumentException("Illegal sub argument #1: '"
+ sendOrReturn
+ "'.  Must be \"send\" or \"return\".");
}
}

/**
 * Corresponds to the getSub Spinner built-in function.
 */
Double getSub(String sendOrReturn) {

assert(logger != null);

// Log this method call
logger.info("getSub(" + sendOrReturn + ");");

if (sendOrReturn == null) {
throw new IllegalArgumentException();
}
else if (sendOrReturn.equals("send")) {
return sendSub;
}
else if (sendOrReturn.equals("return")) {
return returnSub;
}
else {
throw new IllegalArgumentException("Illegal sub argument #1: '"
+ sendOrReturn
+ "'.  Must be \"send\" or \"return\".");
}
}

/**
 * Corresponds to the setSpinMult(discNum, ...) Spinner built-in function.
 */
void setSpinMult(int discNum, double spinMult) {

assert(logger != null);

// Log this method call
logger.info("setSpinMult(" + discNum + ", " + spinMult + ");");

int discIndex = convertDiscNumToIndex(discNum);

outputSpinScriptDisc("spinmult", discNum, spinMult);

discs[discIndex].setSpinMult(spinMult);
}

/**
 * Corresponds to the setSpinMult(all, ...) Spinner built-in function.
 */
void setSpinMultAll(double spinMult) {
```

```
assert(logger != null);

// Log this method call
logger.info("setSpinMultAll(" + spinMult + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

outputSpinScript("spinmult", "all", spinMult);

for (int i = 0; i < discs.length; ++i) {
discs[i].setSpinMult(spinMult);
}
}

/**
 * Corresponds to the getSpinMult Spinner built-in function.
 */
Double getSpinMult(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getSpinMult(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getSpinMult();
}

/**
 * Corresponds to the setSampleStart(discNum, ...) Spinner built-in function.
 */
void setSampleStart(int discNum, double sampleStart) {

assert(logger != null);

// Log this method call
logger.info("setSampleStart(" + discNum + ", " + sampleStart + ");");

int discIndex = convertDiscNumToIndex(discNum);

outputSpinScriptDisc("sampstart", discNum, sampleStart);

discs[discIndex].setSampleStart(sampleStart);
```

```
}

/**
 * Corresponds to the setSampleStart(all, ...) Spinner built-in function.
 */
void setSampleStartAll(double sampleStart) {

assert(logger != null);

// Log this method call
logger.info("setSampleStartAll(" + sampleStart + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

outputSpinScript("sampstart", "all", sampleStart);

for (int i = 0; i < discs.length; ++i) {
discs[i].setSampleStart(sampleStart);
}
}

/**
 * Corresponds to the getSampleStart Spinner built-in function.
 */
Double getSampleStart(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getSampleStart(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getSampleStart();
}

/**
 * Corresponds to the setLength(discNum, ...) Spinner built-in function.
 */
void setLength(int discNum, double length) {

assert(logger != null);

// Log this method call
```

```
logger.info("setLength(" + discNum + ", " + length + ");");

int discIndex = convertDiscNumToIndex(discNum);

outputSpinScriptDisc("len", discNum, length);

discs[discIndex].setSampleLength(length);
}

/**
 * Corresponds to the setLength(all, ...) Spinner built-in function.
 */
void setLengthAll(double length) {

assert(logger != null);

// Log this method call
logger.info("setLengthAll(" + length + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

outputSpinScript("len", "all", length);

for (int i = 0; i < discs.length; ++i) {
discs[i].setSampleLength(length);
}
}

/**
 * Corresponds to the getLength Spinner built-in function.
 */
Double getLength(int discNum) {

assert(logger != null);

// Log this method call
logger.info("getLength(" + discNum + ");");

int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].getSampleLength();
}

/**
```

```
 * Corresponds to the setFile(discNum, ...) Spinner built-in function.
 */
void setFile(int discNum, String file) {

assert(logger != null);

// Log this method call
logger.info("setFile(" + discNum + ", " + file + ");");

int discIndex = convertDiscNumToIndex(discNum);

outputSpinScriptDisc("readfile", discNum, file);

discs[discIndex].setFile(file);
}

/**
 * Corresponds to the setFile(all, ...) Spinner built-in function.
 */
void setFileAll(String file) {

assert(logger != null);

// Log this method call
logger.info("setFileAll(" + file + ");");

if (discs == null) {
throw new NullPointerException("Must call createDiscs first.");
}

outputSpinScript("readfile", "all", file);

for (int i = 0; i < discs.length; ++i) {
discs[i].setFile(file);
}
}

/**
 * Corresponds to the isFileSet Spinner built-in function.
 */
boolean isFileSet(int discNum) {

assert(logger != null);

// Log this method call
logger.info("isFileSet(" + discNum + ");");
```

```
int discIndex = convertDiscNumToIndex(discNum);

return discs[discIndex].isFileSet();
}

void setDisableOutputBuffering(boolean bDisable) {
bDisableOutputBuffering = bDisable;
}

void outputComment(String msg1) {
// Log this method call
logger.info("outputComment(" + msg1 + ");");

outputSpinScriptComment(msg1);
}

void outputComment(String msg1, Double arg1) {
// Log this method call
logger.info("outputComment(" + msg1 + ", " + arg1 + ");");

outputSpinScriptComment(msg1 + arg1.toString());
}

void outputComment(Double arg1, String msg1) {
// Log this method call
logger.info("outputComment(" + arg1 + ", " + msg1 + ");");

outputSpinScriptComment(arg1.toString() + msg1);
}

void outputComment(Double arg1, String msg1, Double arg2) {
// Log this method call
logger.info("outputComment(" + arg1 + ", " + msg1 + ", " + arg2 + ");");

outputSpinScriptComment(arg1.toString() + msg1 + arg2.toString());
}

// ********************************************************
// Helper Functions
// ********************************************************

int convertDiscNumToIndex(int discNum) {

int discIndex = discNum - 1;
```

```
if (discs == null)
throw new NullPointerException("Must call createDiscs first.");

if (discIndex < 0 || discIndex >= discs.length)
throw new ArrayIndexOutOfBoundsException("Invalid disc index '"
+ discNum
+ "'.  Valid values are 1 to "
+ discs.length
+ ".");
return discIndex;
}

/**
 * Builds a SPINscript command string suitable for use by outputSpinScript().
 *
 * SPINscript command format is
 * <prefix>, [discNum|master|all-]<SPINscript Command> <args>;
 *
 * Here is an example:
 * 1, all-gain 0.0;
 *
 * Note: <prefix> is always 1
 *
 * @param spinScriptCmd SPINscript command. Ex: "gain"
 * @param discRef disc reference number or id.  Depending on the
 * command, this can be "all", "master", or a number
 * between 1 and the number of discs (inclusive)
 *  specified by the createDiscs() method call.
 * @param args SPINscript command arguments
 *
 * @see "Spinner Language Reference Manual (LRM)"
 */
String createSpinScriptCmdString( String spinScriptCmd,
String discRef,
Object[] args)
{
// Prefix is the current simulation time
String out = "1"; //Integer.toString((int) getTime());
out += ", ";

if (discRef != null) {
assert(!discRef.equals("0"));
out += discRef + "-";
}

out += spinScriptCmd;
```

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

out += ";";

return out;
}

/**
 * Outputs a SPINscript command string to the outSPINScript PrintStream.
 *
 * Multiple wait commands ("--wait--") in sequence are aggregated and
 * only one wait command is output.  This is done by adding the wait
 * time values instead of outputing anything.  When a non-wait command
 * is output, which should *always* be the case since every SPINscript
 * must end with "--end--", only then is the aggregated wait command
 * output.
 *
 * @see #createSpinScriptCmdString
 */
void outputSpinScript(String spinScriptCmd, String discRef, Object[] args) {
assert(outSPINscript != null);

if (!bOutputFirstTimeComment) {
outputSpinScriptComment("Time: " + Integer.toString((int) getTime()));
bOutputFirstTimeComment = true;
}

if (spinScriptCmd.equals("--wait--")) {
assert(args.length == 1);
accumulatedWaitTime += ((Integer) args[0]).doubleValue();
//flushSpinScriptOutput(); // Uncomment this to show every wait(1)
}
else {
flushSpinScriptOutput();

outSPINscript.println(
createSpinScriptCmdString(spinScriptCmd, discRef, args));

if (bDisableOutputBuffering)
outSPINscript.flush();
}
}
```

```
/**
 * Outputs a SPIN script comment.  Useful for debugging.
 */
void outputSpinScriptComment(String comment) {
outSPINscript.println("1, -------------- " + comment + ";");
if (bDisableOutputBuffering)
outSPINscript.flush();
}

/**
 * Outputs a wait command with the aggregated wait time.  This should never
 * need to be called directly, only from outputSpinScript(), because this
 * gets called automatically when any non-wait command, including the
 * required "--end--" command, is output.
 */
void flushSpinScriptOutput() {

if (accumulatedWaitTime > 0.0) {

Object[] args = new Object[] { new Integer((int) accumulatedWaitTime)};

outSPINscript.println(createSpinScriptCmdString("--wait--", null, args));
if (bDisableOutputBuffering)
outSPINscript.flush();

outputSpinScriptComment("Time: " + Integer.toString((int) getTime()));

accumulatedWaitTime = 0.0;
}
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScript(String spinScriptCmd, String discRef, double arg1) {
Object[] args = new Object[] { new Double(arg1)};
outputSpinScript(spinScriptCmd, discRef, args);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScript(String spinScriptCmd, String discRef, String arg1) {
Object[] args = new Object[] { arg1 };
outputSpinScript(spinScriptCmd, discRef, args);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScript( String spinScriptCmd,
String discRef,
double arg1,
double arg2) {
Object[] args = new Object[] { new Double(arg1), new Double(arg2)};
outputSpinScript(spinScriptCmd, discRef, args);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScriptDisc( String spinScriptCmd,
int discNum,
Object[] args) {
outputSpinScript(spinScriptCmd, Integer.toString(discNum), args);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScriptDisc(String spinScriptCmd, int discNum, double arg1) {
outputSpinScript(spinScriptCmd, Integer.toString(discNum), arg1);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScriptDisc( String spinScriptCmd,
int discNum,
double arg1,
double arg2) {
outputSpinScript(spinScriptCmd, Integer.toString(discNum), arg1, arg2);
}

/** @see #outputSpinScript(String spinScriptCmd, String discRef, Object[] args) */
void outputSpinScriptDisc(String spinScriptCmd, int discNum, String arg1) {
Object[] args = new Object[] { arg1 };
outputSpinScript(spinScriptCmd, Integer.toString(discNum), args);
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {

Simulator sim = new Simulator(System.out, "Simulator.log");

// These Spinner commands should create the "cue-01" SPINscript
sim.createDiscs(6);
// Actual commands from "cue-01":
sim.setGainAll(0.0); // 1, all-gain 0.0;
sim.play(25); // 1, --wait-- 25;
sim.setGainMaster(0.3); // 1, master-gain 0.3;
sim.setReverb("send", 0.11); // 1, reverb-send 0.11;
sim.setReverb("return", 0.1); // 1, reverb-return 0.1;
sim.setSub("send", 0.2); // 1, sub-send 0.2;
sim.setSlideAll("up", 0); // 1, all-slideup 0;
sim.setSlideAll("down", 0); // 1, all-slidedown 0;
sim.seekAll(0, 0); // 1, all-line 0 0;
sim.seekAll(0, 0); // 1, all-line 0 0;
sim.play(100); // 1, --wait-- 100;
sim.setSpinMultAll(1.0); // 1, all-spinmult 1.;
sim.setSampleStartAll(0.0); // 1, all-sampstart 0.0;
sim.setFileAll("Aeolean-Harp"); // 1, all-readfile Aeolean-Harp;
sim.play(8000); // 1, --wait-- 8000;
sim.setSlideAll("up", 10000); // 1, all-slideup 10000;
sim.setSlideAll("down", 10000); // 1, all-slidedown 10000;
sim.setGainAll(0.7); // 1, all-gain 0.7;
sim.seek(2, 4, 32000);
// 1, 2-line 4 32000 8 32000 10 16000 12 16000 14 16000;
sim.seek(2, 8, 32000);
sim.seek(2, 10, 16000);
sim.seek(2, 12, 16000);
sim.seek(2, 14, 16000);
sim.seek(6, 4, 32000);
// 1, 6-line 4 32000 8 32000 10 16000 12 16000 14 16000;
sim.seek(6, 8, 32000);
sim.seek(6, 10, 16000);
sim.seek(6, 12, 16000);
sim.seek(6, 14, 16000);
sim.play(2000); // 1, --wait-- 2000;
sim.seek(4, 4, 32000);
// 1, 4-line 4 32000 8 30203.980469 10 16000 12 15102 14 16000;
sim.seek(4, 8, 30203.980469);
sim.seek(4, 10, 16000);
sim.seek(4, 12, 15102);
sim.seek(4, 14, 16000);
sim.play(66000); // 1, --wait-- 66000;
sim.seek(1, -2, 32000); // 1, 1-line -2 32000 -10 34000;
sim.seek(1, 10, 34000);
sim.seek(3, -2.5, 40000); // 1, 3-line -2.5 40000 -10.5 32000;
sim.seek(3, -10.5, 32000);
sim.seek(5, -3, 48000); // 1, 5-line -3 48000 -15 32000 -15.5 8000;
sim.seek(5, -15, 32000);
sim.seek(5, -15.5, 8000);
sim.play(800); // 1, --wait-- 800;
sim.setSlideAll("up", 1000); // 1, all-slideup 1000;
sim.setSlideAll("down", 1000); // 1, all-slidedown 1000;
sim.play(88000); // 1, --wait-- 88000;
sim.end(999); // 1, --end-- 999;
```

```
}
}
```

D.5.2 Disc.java

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Disc.java,v 1.13 2003/12/17 00:47:32 me133 Exp $
//
//    File name: Disc.java
//   Created By: Marc Eaddy
//   Created On: November 10, 2003, 1:47 AM
//
//      Purpose: simulates the activities of one disc
//
//  Description: using setters and getter to apply spinner prog
//                      functions, disc tracks state of one disc throughout
//                      simulation
//
//////////////////////////////////////////////////////////////////////
/*
 * $Log: Disc.java,v $
 * Revision 1.13  2003/12/17 00:47:32  me133
 * o Fixed package name (should be spinner.compiler)
 *
 * Revision 1.12  2003/12/17 00:10:37  wmb2013
 * ran formatter
 * cleaned comments to prep for documentation
 *
 * Revision 1.11  2003/12/14 04:51:16  wmb2013
 * package declaration fix
 *
 * Revision 1.10  2003/12/14 04:50:45  wmb2013
 * added timeAtTarget variable to track when setspeed hits target
 * fixed play - was not adjusting position if stepping past
 * changed position calc in play to = speed * time + .5 * accleration * time^2
 *
 * Revision 1.9  2003/12/06 18:48:34  wmb2013
 * added special case for seek when time is NOW
 * returned currentSpeed calculation to prior (time in ms)
 * added special case for setSpeed when rampTime is 0
 * fixed bug in setSpeed that was miscalculating acceleration using input
 * targetSpeed of sec rather than ms
 * changed play() to first adjust speed if accelerating, then calc postition,
 * then check for stepping past conditional (target pos for seek and target
 * speed for setSpeed)
 * verified line and speed commands do not interfere with one another
 *
 * Revision 1.8  2003/12/06 16:00:21  wmb2013
 * corrected corrupt package declaration
 *
 * Revision 1.7  2003/12/06 07:10:22  wmb2013
 * seek() changed to calc speed in revs/sec rather than revs/ms
 * play() changed use of time so position calculated correctly
 * play() added redundant check for target position so speed at end of method reports corrrectly
 *
 * Revision 1.6  2003/12/02 05:07:54  me133
 * o Now calls Main.main()
 *
 * Revision 1.5  2003/12/01 06:11:41  me133
 * o Comment was incorrect
 *
 * Revision 1.4  2003/11/19 22:43:42  me133
 * o Added comment
 *
 * Revision 1.3  2003/11/17 01:30:41  me133
 * o Now returns null as per the LRM
 *
 * Revision 1.2  2003/11/11 06:30:39  me133
 * o Log section shouldn't use javadoc style comments
 * o Added comments
 * o No longer parses "dir" variable (uses SlideUp and SlideDown instead)
 * o Organized methods better
 *
 * Revision 1.1  2003/11/10 23:28:51  me133
 * o Created
 */

package spinner.compiler;

import java.io.PrintStream;
import java.util.logging.Logger;

public class Disc {

   // *********************************************************
   // Motion Properties
   // *********************************************************

   public double currentSpeed = 0.0;
   public double currentPosition = 0.0;
   public double currentTime = 0.0;
   public double acceleration = 0.0;
   public double timeAtTarget = 0.0;

   public Double slideUp = null;
   public Double slideDown = null;

   public boolean isSeeking = false;
   public double targetPosition = 0.0;
   public double targetSpeed = 0.0;

   // **********************************************************
   // Sound/Sample Properties
   // **********************************************************

   public Double gain = null;
   public Double spinMult = null;
   public Double sampleStart = null;
   public Double sampleLength = null;
   public String file = null;

   // **********************************************************
   // Misc Properties
   // **********************************************************

   private int discNum = -1;
   private PrintStream outSPINscript = null;
   private Logger logger = null;

   // **********************************************************
   // CTOR
   // **********************************************************

   /** Creates a new instance of Disc */
   public Disc(int discNum, PrintStream outSPINscript, Logger logger) {
      assert(outSPINscript != null);
      assert(logger != null);
      assert(discNum >= 0);

      this.discNum = discNum;
      this.outSPINscript = outSPINscript;
      this.logger = logger;
   }

   // **********************************************************
   // Disc Motion Functions
   // **********************************************************
```

```
/**
 * Corresponds to the seek(discNum, ...) Spinner built-in function.
 *
 * @param seekPosition number of revolutions from the current position
 * @param time time to complete those revolutions in ms
 */
void seek(double seekPosition, double time) {

   assert(time >= 0.0);

   // seek commands are absolute.  They replace previous motion.
   acceleration = 0;

   targetPosition = seekPosition;
   if (time == 0) { //special case  go there NOW
      currentSpeed = 0;
      currentPosition = seekPosition;
      isSeeking = false;
   } else {
      currentSpeed = (targetPosition - currentPosition) / time;
      isSeeking = true;
   }
}

/**
 * Corresponds to the setSpeed(discNum, ...) Spinner built-in function.
 * @param targetSpeed  target speed to achieve in rev/SEC
 * @param rampTime  time in MS to acheive the targetSpeed
 */
void setSpeed(double targetSpeed, double rampTime) {

   assert(rampTime >= 0.0);

   // setSpeed commands are absolute.  They replace previous motion.
   // However, we use whatever current speed we had coming in to the
   // setSpeed call.

   isSeeking = false;

   if (rampTime == 0) { //instantanious speed change
      this.currentSpeed = targetSpeed / 1000;
      this.targetSpeed = 0;
      acceleration = 0;
   } else {
      this.targetSpeed = targetSpeed / 1000;
      //adjust targetspeed to revs/MS
      acceleration = (this.targetSpeed - currentSpeed) / rampTime;
      timeAtTarget = currentTime + rampTime;
   }
}

/**
 * Corresponds to the getSpeed Spinner built-in function.
 */
double getSpeed() {
   return currentSpeed * 1000;
}

/**
 * Corresponds to the setSlide(discNum, "up", ...) Spinner built-in function.
 */
void setSlideUp(double slideUp) {
   this.slideUp = new Double(slideUp);
}

void setSlideDown(double slideDown) {
   this.slideDown = new Double(slideDown);
}

/**
 * Corresponds to the getSlide(discNum, "up") Spinner built-in function.
 */
Double getSlideUp() {
   return slideUp;
}

/**
 * Corresponds to the getSlide(discNum, "down") Spinner built-in function.
 */
Double getSlideDown() {
   return slideDown;
}

/**
 * Corresponds to the getPosition Spinner built-in function.
 */
double getPosition() {
   return currentPosition;
}

// *******************************************************
// Simulation Functions
// *********************************************************

/**
 * Corresponds to the wait Spinner built-in function.
 * Needs work to handle errors related to loss of precision
 * [wmb] could use some optimization
 */
void play(double time) {

   assert(time > 0);

   currentPosition += (currentSpeed * time)
      + (.5 * acceleration * (time * time));
   currentSpeed += acceleration * time;
   currentTime += time;

   //make sure speed is reported correctly after position update if target found
   if (isSeeking) {
      assert(acceleration == 0);
      //check for forward or reverse motion
      if (currentSpeed > 0) { // moving clockwise
         if (currentPosition >= targetPosition) { // at or past target
            currentPosition = targetPosition;
            //so reset current position
            currentSpeed = 0.0;
            isSeeking = false; //stop seeking
         }
      } else { //moving counterclockwise
         if (currentPosition <= targetPosition) { // at or past target
            currentPosition = targetPosition;
            //so reset current position
            currentSpeed = 0.0;
            isSeeking = false; //stop seeking
         }
      }
   }
   //clean up if stepped past speed targets
   //DO NOT check for !isSeeking (assert ok)
   if (acceleration < 0) { // deceleration
      assert(!isSeeking);
      if (currentSpeed <= targetSpeed) { //went too far
         currentSpeed -= acceleration * time; //reset to s0
         currentPosition -= (currentSpeed * time)
            + (.5 * acceleration * (time * time));
         //reset to pos at t0
         currentPosition
```

```
            += (currentSpeed * (timeAtTarget - (currentTime - time)))
            + (.5
               * acceleration
               * (timeAtTarget - (currentTime - time))
               * (timeAtTarget - (currentTime - time)));
         acceleration = 0;
         currentSpeed = targetSpeed;
         currentPosition += currentSpeed * (currentTime - timeAtTarget);
      }
   } else {
      if (acceleration > 0) { //acceleration
         if (currentSpeed > targetSpeed) { //went too far
            currentSpeed -= acceleration * time; //reset to s0
            currentPosition -= (currentSpeed * time)
               + (.5 * acceleration * (time * time));
            //reset to pos at t0
            currentPosition
               += (currentSpeed * (timeAtTarget - (currentTime - time)))
               + (.5
                  * acceleration
                  * (timeAtTarget - (currentTime - time))
                  * (timeAtTarget - (currentTime - time)));
            acceleration = 0;
            currentSpeed = targetSpeed;
            currentPosition += currentSpeed * (currentTime - timeAtTarget);
         }
      }
   }
}

// ************************************************************
// Sound/Sample Functions (no affect on motion)
// ************************************************************

/**
 * Corresponds to the setGain(discNum, ...) Spinner built-in function.
 */
void setGain(double gain) {
   this.gain = new Double(gain);
}

/**
 * Corresponds to the getGain Spinner built-in function.
 */
Double getGain() {
   return gain;
```

```
}

/**
 * Corresponds to the setSpinMult(discNum, ...) Spinner built-in function.
 */
void setSpinMult(double spinMult) {
   this.spinMult = new Double(spinMult);
}

/**
 * Corresponds to the getSpinMult Spinner built-in function.
 */
Double getSpinMult() {
   return spinMult;
}

/**
 * Corresponds to the setSampleStart(discNum, ...) Spinner built-in function.
 */
void setSampleStart(double sampleStart) {
   this.sampleStart = new Double(sampleStart);
}

/**
 * Corresponds to the getSampleStart Spinner built-in function.
 */
Double getSampleStart() {
   return sampleStart;
}

/**
 * Corresponds to the setLength(discNum, ...) Spinner built-in function.
 */
void setSampleLength(double sampleLength) {
   assert(sampleLength > 0);
   this.sampleLength = new Double(sampleLength);
}

/**
 * Corresponds to the getLength Spinner built-in function.
 */
Double getSampleLength() {
   return sampleLength;
}

/**
```

```
     * Corresponds to the setFile(discNum, ...) Spinner built-in function.
     */
    void setFile(String file) {
       this.file = file;
    }

    /**
     * Corresponds to the isFileSet Spinner built-in function.
     */
    boolean isFileSet() {
       return file != null;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
       Main.main(args);
    }
 }
```

D.6 Other

D.6.1 Environment.java

```
/////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Environment.java,v 1.21 2003/12/17 13:42:10 me133 Exp
//
//    File name: Environment.java
//   Created By: Sangho Shiin
//   Created On: November 17, 2003
//
//      Purpose:
//
//  Description:
//
/////////////////////////////////////////////////////////////////////////
/*
 * $Log: Environment.java,v $
 * Revision 1.21  2003/12/17 13:42:10  me133
 * o SpinnerFunction now takes a SpinnerAST for the body
 * o When in debug mode we now output error messages using stdout so that they are intermixed with the r
 *
 * Revision 1.20  2003/12/17 05:27:30  me133
 * o Added extra parallel helper methods
 *
 * Revision 1.19  2003/12/17 04:21:11  ss2020
 * Added hasError().
 *
 * Revision 1.18  2003/12/17 02:21:44  me133
 * o Now keeps track of the parallel nesting level
 *
 * Revision 1.17  2003/12/14 06:03:17  ss2020
 * Changed error handling routine.
 *
 * Revision 1.15  2003/12/11 14:59:44  me133
 * o Moved SymbolTable and VarDef into SymbolTable.java.  I needed to be able to access SymbolTable fron
 *
 * Revision 1.14  2003/12/02 02:12:51  me133
 * o Added ability to get and set array values
 * o Added error checking for setting array values
 * o Now passes variable name for type mismatch error
 * o Fixed array bounds checking
 * o Fixed bug where array value wasn't being returned
 * o Arrays start with 1 not 0
 * o Added args to a few error messages
 *
 * Revision 1.13  2003/12/01 07:00:36  me133
 * o Now supports scoping for parallel branches
 *
 * Revision 1.12  2003/12/01 06:15:07  me133
 * o Added static flag for turning off debug messages.  This is needed if you want to have unadulterated
 * o "Missing function definition" error message now includes function name
 * o Now puts error message args in single quotes
 *
 * Revision 1.11  2003/11/30 21:49:17  me133
 * o Removes <lineNumber> from the error message if its not provided
 * o Capitalized some of the error messages
 *
 * Revision 1.10  2003/11/30 21:02:56  me133
 * o Now uses constants from SpinnerWalker
 * o init() needs to clear all the symbol tables because the Interpreter is going to re-declare/re-regis
 * o Add flag to getFunction() to tell it not to check parameter types.  This is used by the Interprete
 * o Argument mismatch error message are now very verbose, like Java.  Uses new typesAreEqual() functio
 * o Fixed bug in arg mismatch logic that throws a null ptr exception if the declared function paramete
 * o addFunction() now takes parameter names.  This is needed when registering function parameters when
 * o Commented out some diagnostic messages
 * o Left original throwError() method so I could compile.  I'll remove it later.
 * o Error messages are now output to standard error.  Previously, some were output to strerr and some t
 * o Got rid of UNDEFINED.  Just use INIT instead (means same thing).
 * o Added paramsToString() and paramToString() to help when creating error messages
 * o Had problems with SymbolTable implementation that was causing exceptions to be thrown.  I made a fe
 *
 * Revision 1.7  2003/11/24 04:41:27  me133
 * o Now handles case when argument list is empty (previously this caused a null ptr exception)
 * o Added back main() method (this may have been removed during the merge process)
 * o Commented out some debug statements so it doesn't get intermixed with the compiler output
 *
 * Revision 1.6  2003/11/24 00:03:21  ss2020
 *
 * Added function argument checks
 *
 * Sangho,
 *
 * Revision 1.5  2003/11/23 21:07:14  me133
 * o Commented out some print statements.  You can put these back if you want.
 * o Added main()
 *
 * Revision 1.4  2003/11/23 20:38:51  me133
 * o Message now includes function name
```

```
 *
 * Revision 1.3  2003/11/23 09:52:28  me133
 * o Replaced FuncDef with SpinnerFunction.  SpinnerFunction can execute builtin or user-defined functio
 * o Builtin function table is now built up by the Main class using a SpinnerFunction helper
 * o Changed funcCheck() to getFunction() so that we can execute the function immediately after if we wa
 * o addFunction() now handles adding functions with more than one parameter
 * o Commented out getFuncBody().  You should call getFunction().invoke() instead.
 *
 * Revision 1.2  2003/11/18 14:57:21  ss2020
 * Added getFuncBody(). This function returns the AST of the funtion body.
 *
 * Revision 1.1  2003/11/17 07:38:35  me133
 * o Initial checkin
 */

package spinner.compiler;

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

public class Environment {

    public static boolean bDebug = true;

    public final static int ERROR = -1;
    public final static int OK = 0;

    public static final int nINIT_TYPE = SpinnerWalker.nINIT_TYPE;
    public static final int nDOUBLE_TYPE = SpinnerWalker.nDOUBLE_TYPE;
    public static final int nBOOLEAN_TYPE = SpinnerWalker.nBOOLEAN_TYPE;
    public static final int nASSIGNMENT_TYPE = SpinnerWalker.nASSIGNMENT_TYPE;
    public static final int nVOID_TYPE = SpinnerWalker.nVOID_TYPE;
    public static final int nARRAY_DOUBLE_TYPE
        = SpinnerWalker.nARRAY_DOUBLE_TYPE;
    public static final int nARRAY_BOOLEAN_TYPE
        = SpinnerWalker.nARRAY_BOOLEAN_TYPE;
    public static final int nSTRING_TYPE = SpinnerWalker.nSTRING_TYPE;

    HashMap builtInFuncTable;
    HashMap userDefFuncTable;
    SymbolTable currentSymbolTable;
    SymbolTable mainSymbolTable;
    SymbolTable tempSymbolTable;
    boolean useSymbolTable;
    SemanticErrorHandler errorHandler;
int nParallelBranchNestingLevel = 0;
```

```
    boolean haveErrors;

    /* SymbolTable()
     * Consturctor
     */
    Environment() {
        builtInFuncTable = new HashMap();
        userDefFuncTable = new HashMap();
        mainSymbolTable = new SymbolTable(null, this, false);
        currentSymbolTable = mainSymbolTable;
        useSymbolTable = false;
        errorHandler = new SemanticErrorHandler();
haveErrors = false;
    }

    /* init()
     * Initialize Environment. When the backend use this symbol table,
     * this function must be called.
     */
    public void init() {
        mainSymbolTable = new SymbolTable(null, this, false);
        currentSymbolTable = mainSymbolTable;
        useSymbolTable = true;
    }

    /* addVariable()
     * Add variable to symbol table.
     */
    public void addVariable(String varName, int type, String value) {
        currentSymbolTable.addVariable(varName, type, value);
    }

    /* addVariable()
     * Add variable to symbol table.
     */
    public void addVariable(SpinnerAST varAST, int type, String value) {
        currentSymbolTable.addVariable(varAST, type, value);
    }

    /* addVariable()
     * Add variable to symbol table.
     */
    public void addVariable(String varName, int type, String size,
                            String value) {
        currentSymbolTable.addVariable(varName, type, size, value);
    }
```

```
/* addVariable()
 * Add variable to symbol table.
 */
public void addVariable(SpinnerAST varAST, int type, String size,
                        String value) {
    currentSymbolTable.addVariable(varAST, type, size, value);
}

/* varCheck()
 * Checks if the variable is declared already.
 */
public int varCheck(SpinnerAST varAST, int type) {
    return currentSymbolTable.varCheck(varAST, type);
}

/* varCheck()
 * Checks if the variable is declared already.
 */
public int varCheck(String varName) {
    return currentSymbolTable.varCheck(varName);
}

/* varCheck()
 * Checks if the variable is declared already.
 */
public int varCheck(SpinnerAST varAST) {
    return currentSymbolTable.varCheck(varAST);
}

/* getType()
 * Returns the type of the symbol
 */
public int getType(String varName) {
    return currentSymbolTable.getType(varName);
}

/* getDoubleValue()
 * Returns the value of double type variable.
 */
public double getDoubleValue(String varName) {
    return currentSymbolTable.getDoubleValue(varName);
}

/* getDoubleValue()
 * Returns the value of double type variable.
```

```
     */
    public double getDoubleValue(String varName, int nIndex) {
        return currentSymbolTable.getDoubleValue(varName, nIndex);
    }

    /* getBooleanValue()
     * Returns the value of boolean type variable.
     */
    public boolean getBooleanValue(String varName) {
        return currentSymbolTable.getBooleanValue(varName);
    }

    /* getBooleanValue()
     * Returns the value of boolean type variable.
     */
    public boolean getBooleanValue(String varName, int nIndex) {
        return currentSymbolTable.getBooleanValue(varName, nIndex);
    }

    /* setValue()
     * Sets the value of a variable.
     */
    public void setValue(String varName, String value) {
        currentSymbolTable.setValue(varName, value);
    }

    /* setValue()
     * Sets the value of an array item.
     */
    public void setValue(String varName, int nIndex, String value) {
        currentSymbolTable.setValue(varName, nIndex, value);
    }

    /* funcDef()
     * Create a symbol table for the function.
     */
    public void beginFuncDef(String funcName) {
        tempSymbolTable = currentSymbolTable;
        currentSymbolTable = new SymbolTable(tempSymbolTable, this, false);
    }

    /* endFuncDef()
     * Save the symbol table for the function into function defintion.
     */
    public void endFuncDef(String funcName) {
        SpinnerFunction func = getFunction(funcName, null, false);
        if (func != null)
            func.setSymbolTable(currentSymbolTable);
        currentSymbolTable = tempSymbolTable;
    }

    /* funcCheck()
     * Checks if the function is a built-in function or it is defined.
     */
    public SpinnerFunction getFunction(String funcName, int params[],
                                       boolean bCheckParamTypes) {
        SpinnerFunction func = (SpinnerFunction)builtInFuncTable.get(funcName);
        if (func == null) {
            func = (SpinnerFunction)userDefFuncTable.get(funcName);
            if (func == null) {
                throwError(301, funcName);
                return null;
            }
        }

        if (bCheckParamTypes) {

            /* Check params */
            int definedParams[] = func.getParams();

            if (!typeArraysAreEqual(params, definedParams)) {
                String strFound = "\n\tFound : " +
                    funcName + "(" + paramsToString(params) + ")";
                String strRequired = "\n\tRequired : " + funcName + "(" +
                    paramsToString(definedParams) + ")";
                throwError("Argument mismatch for function: " + funcName +
                           strFound + strRequired);
                return null;
            }
        }

        return func;
    }

    /* funcCheck()
     * Checks if the function is a built-in function or it is defined.
     */
    public SpinnerFunction checkFunction(SpinnerAST funcAST, int params[],
                                   boolean bCheckParamTypes) {
        String funcName = funcAST.toString();
        SpinnerFunction func = (SpinnerFunction)builtInFuncTable.get(funcName);
        if (func == null) {
            func = (SpinnerFunction)userDefFuncTable.get(funcName);
            if (func == null) {
                throwError(301, funcName, funcAST);
                return null;
            }
        }

        if (bCheckParamTypes) {

            /* Check params */
            int definedParams[] = func.getParams();

            if (!typeArraysAreEqual(params, definedParams)) {
                String strFound = "\n\tFound : " +
                    funcName + "(" + paramsToString(params) + ")";
                String strRequired = "\n\tRequired : " + funcName + "(" +
                    paramsToString(definedParams) + ")";
                throwError("Argument mismatch for function: " + funcName +
                           strFound + strRequired, funcAST);
                return null;
            }
        }

        return func;
    }

    public static boolean typeArraysAreEqual(int[] a, int[] b)
    {
        if (a == null && b == null)
        {
            return true;
        }
        else if (a == null)
        {
            return b.length == 0;
        }
        else if (b == null)
        {
            return a.length == 0;
        }
        else if (a.length != b.length)
        {
            return false;
        }
        else
        {
            for(int i = 0; i < a.length; ++i) {
                if (a[i] != b[i]) return false;
            }
            return true;
        }
    }

    /* getFunction
     * Checks if the function is a built-in function or it is defined.
     * This function is used by Interpreter.
     */
/*
    public SpinnerFunction getFunction(String funcName) {
        SpinnerFunction func = (SpinnerFunction)builtInFuncTable.get(funcName);
        return func;
    }
 */

    /* addBuiltinFunction()
     * Add builtin functions.
     */
    public void addBuiltinFunction(int retType, String funcName,
                                   int[] paramTypes) {
        assert(funcName != null);
        assert(funcName.length() > 0);

        SpinnerFunction func =
            new SpinnerFunction(retType, funcName, paramTypes);
        builtInFuncTable.put(funcName, func);
    }

    /* addFunction()
     * Add function definition to symbol table.
     */
    public void addFunction(int type, String funcName, int paramTypes[], String[] paramNames,
                            SpinnerAST node) {
        if (builtInFuncTable.get(funcName) != null) {
            throwError(306, funcName);
            return;
        }
        SpinnerFunction func = new SpinnerFunction(type, funcName, paramTypes, paramNames,
                                                   node);
        userDefFuncTable.put(funcName, func);
        debug(funcName + " is added into UDF table."+node);
    }
    /* enterScope()
     * Creates another symbol table, and adds it to symbolTables, points it
     * as currentSymbolTable.
     */
    public void enterScope() {
        currentSymbolTable = currentSymbolTable.enterScope(useSymbolTable);
        //debug("Enter scope");
    }

    /* enterScope()
     * Changes the current symbol table to be the one specified.  This is
 * needed when implementing parallel branches because the caller has
 * to hold onto the symbol table for each branch.
     */
    public void enterScope(SymbolTable symTable) {
        currentSymbolTable = symTable;
        //debug("Enter scope");
    }

    /* leaveScope()
     * Remove current symbol table, and points the previous(upper) one as
     * current symbol table.
     */
    public void leaveScope() {
        currentSymbolTable = currentSymbolTable.leaveScope();
        //debug("Leave scope");
    }

    /**
     * Called each time a parallel scope is entered.  Must be matched with an
 * equal number of leaveParallel() calls.
     */
public void enterParallel() {
assert(nParallelBranchNestingLevel >= 0);
++nParallelBranchNestingLevel;
}

    /**
     * Called each time a parallel scope is left.  Must be matched with all
 * enterParallel() calls.
     */
public void leaveParallel() {
--nParallelBranchNestingLevel;
assert(nParallelBranchNestingLevel >= 0);
}
```

```
/**
 * Whether we are executing code inside a parallel branch or not
 */
public boolean isInParallelBranch() {
assert(nParallelBranchNestingLevel >= 0);
return nParallelBranchNestingLevel > 0;
}

/**
 * Whether we are executing code in a top-level parallel branch or
 * a nested parallel branch.
 */
public boolean isInTopParallelBranch() {
assert(nParallelBranchNestingLevel >= 0);
return nParallelBranchNestingLevel == 1;
}

/**
 * Returns the current parallel branch nesting level.  Should only be
 * used for debugging purposes.
 */
public int getParallelNestingLevel() {
return nParallelBranchNestingLevel;
}

/**
 * Creates a symbol table that is scoped by the current symbol table.
 * This is used by the Interpreter so that each parallel branch has
 * its own symbol table.  We can then switch back-and-forth by call
 * enterScope(SymbolTable).
 *
 * @see #enterScope(SymbolTable symTable)
 */
public SymbolTable createSymbolTable() {
return new SymbolTable(currentSymbolTable, this, false);
}

    /* enterFunction()
     * Change the current symbol table into the symbol table of the function.
     */
    public void enterFunction(String funcName) {
        tempSymbolTable = currentSymbolTable;
        SpinnerFunction func = getFunction(funcName, null, false);
        if (func != null)
            currentSymbolTable = func.getSymbolTable();
        else
            debug("No funtion???");
        currentSymbolTable.setOuterSymbolTable(tempSymbolTable);
    }

    /* leaveFunction()
     * Change the current symbol table into the symbol table of the function.
     */
    public void leaveFunction() {
        currentSymbolTable = tempSymbolTable;
    }

    /* throwError()
     * Show compile error.
     */
    public void throwError(int errorCode, String arg) {
        haveErrors = true;
        errorHandler.throwError(errorCode, arg);
    }

    /* throwError()
     * Show compile error.
     */
    public void throwError(int errorCode, String arg, String file, int line) {
        haveErrors = true;
        errorHandler.throwError(errorCode, arg, file, line);
    }

    /* throwError()
     * Show compile error.
     */
    public void throwError(String err) {
        haveErrors = true;

if (bDebug)
        System.out.println("<Compile Error>: "+err);
else
        System.err.println("<Compile Error>: "+err);
    }

    /* throwError()
     * Show compile error.
     */
    public void throwError(String err, AST errNode) {
        haveErrors = true;
        errorHandler.throwError(err, errNode);
    }

    /* throwError()
     * Show compile error.
     */
    public void throwError(int errorCode, String arg, AST errNode) {
        haveErrors = true;
        errorHandler.throwError(errorCode, arg, errNode);
    }

    /* debug()
     * Shows debug message
     */
    public void debug(String msg) {
if (bDebug)
        System.out.println(msg);
    }

    public boolean hasError() {
        return haveErrors;
    }

    /* This function is just for debugging.
     * It shows how to use getBooleanValue() and getDoubleValue() in the
     * interpreter.
     */
    public void printValue(String varName) {
        int type = getType(varName);
        if (type != nINIT_TYPE) {
            debug("Variable name: " + varName);
        }

debug("  --> value : " + getDoubleValue(varName) + " " + (type == nBOOLEAN_TYPE ? "(nBOOLEAN_TYPE)" : "
    }

static String paramsToString(int[] params) {
if (params == null) return "";
String strParams = "";
String strParam;
for(int i = 0; i < params.length; ++i) {
if (params[i] == nVOID_TYPE) {
strParam = "<INVALID:void>";
}
else {
strParam = paramToString(params[i]);
}
if (i > 0)
strParams += ", ";

strParams += strParam;
}

return strParams;
}

static String paramToString(int param) {

if (param == nINIT_TYPE) {
return "<INVALID:uninitialized>";
}
else if (param == nDOUBLE_TYPE) {
return "double";
}
else if (param == nBOOLEAN_TYPE) {
return "boolean";
}
else if (param == nVOID_TYPE) {
return "void";
}
else if (param == nSTRING_TYPE) {
return "string";
}
else {
return "<UNKNOWN:" + new Integer(param).toString() + ">";
}
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
Main.main(args);
}
}

class SemanticErrorHandler {

    HashMap errorTable;

    SemanticErrorHandler() {
        errorTable = new HashMap();
    errorTable.put(new Integer(101), new SemanticError(101,
                                       "Double expected <ARG> "));
    errorTable.put(new Integer(102), new SemanticError(102,
                                      "Boolean expected <ARG> "));
    errorTable.put(new Integer(103), new SemanticError(103,
                                      "Type mismatch <ARG> "));
    errorTable.put(new Integer(201), new SemanticError(201,
                        "Variable <ARG> not defined "));
    errorTable.put(new Integer(202), new SemanticError(202,
                        "Variable <ARG> not initialized "));
    errorTable.put(new Integer(203), new SemanticError(203,
                        "Variable <ARG> is null "));
    errorTable.put(new Integer(204), new SemanticError(204,
                        "Variable <ARG> already exists "));
    errorTable.put(new Integer(301), new SemanticError(301,
                        "Missing function definition <ARG> "));
    errorTable.put(new Integer(302), new SemanticError(302,
                        "Missing returnType "));
    errorTable.put(new Integer(303), new SemanticError(303,
                        "Statement not reachable "));
    errorTable.put(new Integer(304), new SemanticError(304,
                    "Function <ARG> returnType mismatch "));
    errorTable.put(new Integer(305), new SemanticError(305,
              "Function definition <ARG> already exists "));
    errorTable.put(new Integer(306), new SemanticError(306,
                        "Function <ARG> is builtIn "));
    errorTable.put(new Integer(307), new SemanticError(307,
                    "The number of arguments for function <ARG> mismatch"));
    errorTable.put(new Integer(308), new SemanticError(308,
                    "Argument type for function <ARG> mismatch"));
    errorTable.put(new Integer(309), new SemanticError(309,
                    "Function <ARG> definition error"));
    errorTable.put(new Integer(310), new SemanticError(310,
                    "Missing return statement in function <ARG>"));
    errorTable.put(new Integer(401), new SemanticError(401,
                        "ArrayIndexOutOfBounds <ARG> "));
    errorTable.put(new Integer(402), new SemanticError(402,
                        "Array <ARG> cannot be void type "));
    errorTable.put(new Integer(403), new SemanticError(403,
                        "Array <ARG> index should be double type "));
    errorTable.put(new Integer(404), new SemanticError(404,
                    "Array <ARG> size must be a positive non-zero number"));
};

public void throwError(int errorCode, String arg) {
        SemanticError error =
            (SemanticError)errorTable.get(new Integer(errorCode));
        if (error != null)
            error.printError(arg);
    }

    public void throwError(int errorCode, String arg, String file, int line) {
        SemanticError error =
            (SemanticError)errorTable.get(new Integer(errorCode));
        if (error != null)
            error.printError(arg, file, line);
    }

    public void throwError(int errorCode, String arg, AST errNode) {
        SemanticError error =
            (SemanticError)errorTable.get(new Integer(errorCode));
        if (error != null)
            error.printError(arg, errNode);
    }

    public void throwError(String error, AST errNode) {
        SpinnerAST node = (SpinnerAST)errNode;
        int line = node.getLine();
        String file = node.getFileName();
        while(line == 0) {
            node = (SpinnerAST)errNode.getFirstChild();
            line = node.getLine();
        }

if (Environment.bDebug)
        System.out.println(file+":"+line+": "+error);
else
        System.err.println(file+":"+line+": "+error);
    }
}

class SemanticError {

    int errorCode;
    String errorMessage;

    SemanticError(int code, String message) {
        errorCode = code;
        errorMessage = message;
    }
    public void printError(String arg) {
        String errMsg;
        if (arg != null)
            errMsg = errorMessage.replaceFirst("<ARG>", "'" + arg + "'");
        else
            errMsg = errorMessage.replaceFirst("<ARG>","");

if (Environment.bDebug)
        System.out.println("<Compile Error>: " + errMsg);
else
        System.err.println("<Compile Error>: " + errMsg);
    }

    public void printError(String arg, String file, int line) {
        String errMsg;
        if (arg != null)
            errMsg = errorMessage.replaceFirst("<ARG>", "'" + arg + "'");
        else
            errMsg = errorMessage;

        // We don't have line numbers yet
if (Environment.bDebug)
        System.out.println(file + ":" + line +": "+ errMsg);
else
        System.err.println(file + ":" + line +": "+ errMsg);
    }

    public void printError(String arg, AST astNode) {
        SpinnerAST node = (SpinnerAST)astNode;
        int line = node.getLine();
        String file = node.getFileName();
        String errMsg;
        if (arg != null)
            errMsg = errorMessage.replaceFirst("<ARG>", "'" + arg + "'");
        else
            errMsg = errorMessage.replaceFirst("<ARG>","");

        while(line == 0) {
            node = (SpinnerAST)astNode.getFirstChild();
            line = node.getLine();
        }

        // We don't have line numbers yet
if (Environment.bDebug)
        System.out.println(file + ":" + line +": "+ errMsg);
else
```

```
        System.err.println(file + ":" + line +": "+ errMsg);
    }
}
```

D.6.2 Main.java

```
/////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/Main.java,v 1.13 2003/12/17 14:04:26 me133 Exp $
//
//    File name: Main.java
//   Created By: Marc Eaddy
//   Created on: 11/2/2003
//
//      Purpose:
//
//  Description:
//
/////////////////////////////////////////////////////////////////////////
/**
 * $Log: Main.java,v $
 * Revision 1.13  2003/12/17 14:04:26  me133
 * o Added some commented out lines used for testing
 * o Now if debug is true we flush the spin script output after every line intead of buffering it
 *
 * Revision 1.12  2003/12/17 04:22:50  ss2020
 * If there is any error in lexer, parser, or semantic analyzer, then it stop there.
 *
 * Revision 1.11  2003/12/17 02:17:14  me133
 * o Now catches any mistakenly uncaught PausedExceptions
 *
 * Revision 1.10  2003/12/11 15:00:48  me133
 * o Fixed bug that occured when you specify the -debug flag
 *
 * Revision 1.9  2003/12/08 01:17:13  wmb2013
 * added import for java.util.Iterator
 *
 * Revision 1.8  2003/12/07 06:48:07  ss2020
 * Added preprocessor.
 *
 * Revision 1.7  2003/12/01 07:45:35  me133
 * o Can now specify -debug flag to turn on debug output
 *
 * Revision 1.6  2003/12/01 06:12:56  me133
 * o Turns off debug messages by default
 * o Now explicitly calls Simulator.end() to ensure that the last SPINscript command output is "--end--
 *
 * Revision 1.5  2003/11/30 19:54:05  me133
 * o Now uses SpinnerWalker
 *
 * Revision 1.4  2003/11/24 05:46:44  me133
 * o Commented out using the walker until I can properly integrate it.  Sangho, feel free to uncomment t
 *
 * Revision 1.3  2003/11/23 09:49:21  me133
 * o Now uses the SpinnerWalker and Interpreter classes
 *
 * Revision 1.2  2003/11/19 22:51:19  me133
 * o Now invokes interpreter (probably have to remove this later)
 *
 * Revision 1.1  2003/11/02 11:03:06  me133
 * Created
 */

package spinner.compiler;

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

import antlr.*;
import antlr.collections.*;
import antlr.debug.misc.ASTFrame;

public class Main {
    public static void main(String[] args) {

try {
            // Run Lexer
SpinnerLexer lexer = null;

Environment.bDebug = false;
SpinnerPreprocessor sp = null;

String strFileName = null;
for(int i = 0; i < args.length; ++i){
if (args[i].equals("-debug")) {
Environment.bDebug = true;
}
else
{
strFileName = args[i];
}
}

// JUST FOR DEBUGGING!!!!!!  PLEASE REMOVE!!!! - ME
```

```
//strFileName = "tests/testSpeedSeek.spinner";
//Environment.bDebug = true;

if (strFileName != null)
{
if (Environment.bDebug) {
System.out.println("SPINNER: Compiling " + strFileName + "...");
}
sp = new SpinnerPreprocessor(strFileName);
}
else
{
  //lexer = new SpinnerLexer(new DataInputStream(System.in));
  return;
}

// Combine all the ASTs into one AST.
Iterator iter = sp.getAST().iterator();
if (!iter.hasNext()) {
    return;
}
CommonAST parseTree = (CommonAST)iter.next();
while(iter.hasNext()) {
  CommonAST ast = (CommonAST)iter.next();
  parseTree.addChild(ast.getFirstChild());
}

if (Environment.bDebug) {
System.out.println(parseTree.toStringList());
}

            // Run symantic analyzer
            SpinnerWalker walker = new SpinnerWalker();

            // Get environment from semantic analyzer.
Environment env = new Environment();
env.init();
SpinnerFunction.addBuiltins(env);
            walker.setEnv(env);

int nWalkerStatus = walker.expr(parseTree);
if (env.hasError()) {
    return;
}
if (nWalkerStatus == env.OK)
{
```

```
// Run Interpreter
SpinnerInterpreter interpreter = new SpinnerInterpreter();

Simulator sim = new Simulator(System.out, "Simulator.log");

sim.setDisableOutputBuffering(Environment.bDebug);

env.init();

interpreter.setEnvironment(env);
interpreter.setSimulator(sim);

try {
interpreter.expr(parseTree);
}
catch (PausedException ex) {
System.err.println("Unexpected Error: Paused exception thrown outside of a parallel statement.");
ex.printStackTrace();
}

sim.end(999);
}

if (Environment.bDebug) {
ASTFrame frame = new ASTFrame("The tree", parseTree);
frame.setVisible(true);
}

} catch(Exception e) {
e.printStackTrace();
        }
    }
}
```

D.6.3 SpinnerRet.java

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/SpinnerRet.java,v 1.3 2003/12/17 16:07:10 me133 Exp $
//
//    File name: SpinnerRet.java
//   Created By: Marc Eaddy
//   Created On: December 8, 2003, 6:14 PM
//
//      Purpose: Holds return values for Spinner expressions
//
//  Description: Encapsulates functionality for dealing with return
// values returned by executing Spinner expressions.
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: SpinnerRet.java,v $
 * Revision 1.3  2003/12/17 16:07:10  me133
 * o Comments
 *
 * Revision 1.2  2003/12/17 13:46:21  me133
 * o Now safeCompareTo is *really* safe.  Before it would throw a null ptr exception if the values were
 *
 * Revision 1.1  2003/12/17 02:24:13  me133
 * o Created
 */

package spinner.compiler;

/**
 * Holds return values for Spinner expressions.  Encapsulates functionality
 * for dealing with return values returned by executing Spinner expressions.
 */
public class SpinnerRet {

// ---------------------------------------------------
// DATA
// ---------------------------------------------------

// Useful constants
public static SpinnerRet retVOID = new SpinnerRet();
public static SpinnerRet retTRUE = new SpinnerRet(1.0);
public static SpinnerRet retFALSE = new SpinnerRet(0.0);
```

```
/**
 * Holds the real value of the expression
 */
Double val = null;

// --------------------------------------------------
// CTOR
// --------------------------------------------------

public SpinnerRet() {
val = null;
}

public SpinnerRet(Double val) {
if (val == null)
this.val = null;
else
this.val = new Double(val.doubleValue());
}

public SpinnerRet(double val) {
this.val = new Double(val);
}

public SpinnerRet(String val) {
this.val = new Double(val);
}

public SpinnerRet(boolean val) {
this.val = val ? new Double(1.0) : new Double(0.0);
}

// --------------------------------------------------
// METHODS
// --------------------------------------------------

/**
 * Wraps Double.doubleValue()
 */
public double doubleValue() {
assert(val != null);
return val.doubleValue();
}

/**
```

```
 * Wraps Double.intValue()
 */
public int intValue() {
assert(val != null);
return val.intValue();
}

/**
 * true if val == 1 (val must not be null)
 */
public boolean boolValue() {
return !isFalse();
}

/**
 * true if val == 1 (val must not be null)
 */
public boolean isTrue() {
return !isFalse();
}

/**
 * true if val == 0 (val must not be null)
 */
public boolean isFalse() {
assert(val != null);
return val.doubleValue() == 0;
}

/**
 * true if val is null
 */
public boolean isVoid() {
return val == null;
}

/**
 * true if val is null
 */
public boolean isNull() {
return isVoid();
}

/**
 * compares two SpinnerRet objects
 */
```

```
public boolean equals(SpinnerRet rhs) {
return safeCompareTo(this, rhs) == 0;
}

/**
 * compares two SpinnerRet objects.  Can handle converting from
 * Object or Double to SpinnerRet.
 */
public boolean equals(Object rhs) {
if (rhs instanceof SpinnerRet)
{
return equals((SpinnerRet) rhs);
}
else if (rhs instanceof Double)
{
return equals(new SpinnerRet((Double) rhs));
}
else
{
return false;
}
}

/**
 * string representation.  null is "null".
 */
public String toString() {
if (val == null)
return "null";
else
return val.toString();
}

/**
 * Changes val from 1 to 0 or from 0 to 1.
 * @returns new value (this)
 */
public SpinnerRet not() {
if (val != null)
{
val = val.doubleValue() != 0 ? new Double(0) : new Double(1);
}

return this;
}
```

```
/**
 * Switches the sign of the value
 * @returns new value (this)
 */
public SpinnerRet switchSign() {
if (val != null)
{
val = new Double(-val.doubleValue());
}

return this;
}

/**
 * Increments value
 * @returns new value (this)
 */
public SpinnerRet increment() {
if (val != null)
{
val = new Double(val.doubleValue() + 1);
}
return this;
}

/**
 * Decrements value
 * @returns new value (this)
 */
public SpinnerRet decrement() {
if (val != null)
{
val = new Double(val.doubleValue() - 1);
}
return this;
}

/**
 * Converts boolean to retTRUE or retFALSE
 */
public static SpinnerRet boolToRet(boolean b) {
return b ? retTRUE : retFALSE;
}

/**
 * Compares two SpinnerRet objects.  Handles nulls.
 */
    public static int safeCompareTo(SpinnerRet lhs, SpinnerRet rhs) {
        if (lhs == null && rhs == null)
            return 0;
        else if (lhs == null)
            return -1;
        else if (rhs == null)
            return 1;
        else if (lhs.val == null && rhs.val == null)
return 0;
else if (lhs.val == null)
return -1;
else if (rhs.val == null)
return -1;
else
            return lhs.val.compareTo(rhs.val);
    }

/**
 * Adds two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeAdd(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.val.doubleValue() + rhs.val.doubleValue());
}

/**
 * Subtracts two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeSubtract(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.val.doubleValue() - rhs.val.doubleValue());
}

/**
 * Multiplies two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeMultiply(SpinnerRet lhs, SpinnerRet rhs) {
```

```
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.val.doubleValue() * rhs.val.doubleValue());
}

/**
 * Divides two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeDivide(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.val.doubleValue() / rhs.val.doubleValue());
}

/**
 * Calculates modulo for two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeMod(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.val.doubleValue() % rhs.val.doubleValue());
}

/**
 * ANDs two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeAnd(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.isTrue() && rhs.isTrue());
}

/**
 * ORs two SpinnerRet objects.  Handles nulls.
 */
public static SpinnerRet safeOr(SpinnerRet lhs, SpinnerRet rhs) {
if (lhs == null || rhs == null)
return null;
else if (lhs.val == null || rhs.val == null)
return null;
else
return new SpinnerRet(lhs.isTrue() || rhs.isTrue());
}

/**
 * NOTs a SpinnerRet object.  Handles nulls.
 */
public static SpinnerRet safeNot(SpinnerRet lhs) {
if (lhs == null)
return null;
else
return new SpinnerRet(!lhs.isTrue());
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
Main.main(args);
}
}
```

D.6.4 Library Functions - DiscLib1.spinner

```
/////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/DiscLib1.spinner,v 1.2 2003/12/05 23:13:30 wmb2
//
//    File name: DiscLib1.spinner
//   Created By: William Beaver
//   Created on: 10/3/2003
//    $Revision: 1.2 $
//
//      Purpose: library for fucntion
//
//  Description: libray file for some spinner functions
//
/////////////////////////////////////////////////////////////////////
/*
 * $Log: DiscLib1.spinner,v $
 * Revision 1.2  2003/12/05 23:13:30  wmb2013
 * corrected bug in cleanup function
 * changed setup to func to only createDiscs of fixed size
 * created getAngle function that returns degrees 0-359
 *
 *
 */



//returns all discs to zero position and sets system values to zero
void func cleanup(){
    seek(all, 0, 2000); //return everyone home in a nice way
    setGain(0.0);
    setReverb(0.0);
    return;
}

void func setup(){
createDiscs(6);
return;
}

//return a discs position in degrees 0-359
double func getAngle(double discNum){
if (getDiscsSize = null) then{ //discs haven't been initialized
return null;
}
else{ // disc initialized...ok
if ((discNum < 0) OR (discNum > getDiscSize()) then{ //discNum reference outside of disc bounds
return null;
}
else { //discNum boundary ok
return (getPostion(discNum)%1)*360;
}
}
}
```

Appendix E

Compiler Tests

(Note: Each SPINNER test program has an equivalent ".expected" SPINscript file. They are not included here.)

E.1 Parser, Lexer and Semantic Analysis

E.1.1 correctLrm.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/correctLrm.spinner,v 1.3 2003/12/02 02:17:26 me
//
//    File name: correctLrm.spinner
//   Created By: Gaurav Khandpur
//   Created on: 11/11/2003
//
//      Purpose: Correct Spinner Program
//
//  Description: Parser should be able to pass the correctLrm.spinner
//                      compeletely without errors.  Went through the LRM and
//                      wrote simple test case for each piece of functionality.
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: correctLrm.spinner,v $
 * Revision 1.3  2003/12/02 02:17:26  me133
 * o Fixed compile error (I made)
 *
 * Revision 1.2  2003/12/01 22:53:09  me133
 * o Minor comment change
 *
 * Revision 1.1  2003/12/01 06:30:26  me133
 * o Created
 *
 */

//testing import;
import ("abc.spin");

//testing comments

/* This is a comment.
*/

//this is also a comment

//testing constant names
double MAX_VALUE:=10;

//testing variable and assignments
double a4rfre332dsds;
boolean we323:=true;
boolean we324:=false;
double a:=null;
double arr1[10];

//array of an array
arr1 arr2[10];

//checking for statements

//empty statement
;
;;

//testing break

{
double test:=8;
if(test<9) then
{
break;
}
if(test>7)then{
   callFunc();
}
}

//testing if then else
if(x<y) then{
callFunc1(param1,param2);
} else {
callFunc2(param1,param2);
}

if(true) then {}
else {a:=5;}

//while..do
while(c=false) do {;}

//repeat..until
repeat{
if(test<5){}
}until(true)

//for statement
for(double i;i<10.0;i++){}

//testing return
void funcCall1(){
a:=5;
b:=6;
return;
}

//testing concurrency operator
//block with block allowed
{a:=5;
 b:=6;
}||
{
c:=7;
d:=6;
}

//two standalone funcs. in parallel allowed
callFunc1(); || callFunc2();

//testing array initialization, probably this is dealt with back end
double arr1[2];
```

```
arr1[1]:=2;
arr1[2]:=3;

//array access

if(b<arr1[2]) then {}

//testing logical and

while(a<b and c<d) do {;}
if (a=b or c!=d) then callFunc1();

//testing postfix incr
x++;
y++;

//testing postfix decr
x--;
y--;

//testing unary operator- not

if(not(false)) then {a:=6;}

//testing multiplicative operator
a:=b*5;
c:=4*6;

//testing divide

a:=b/4;

//testing remainder

c:=a%3;

//testing addition
double arr1[2];
arr1[1]=a+b;
arr1[2]=6-3;

//testing logical operators
if(a<b) then
if(b>c) then
if(c<=d) then
if(d>=e) then
if(e=f) then {;}
```

E.1.2 incorrectLrm.spinner

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/incorrectLrm.spinner,v 1.2 2003/12/01 22:53:09
//
//    File name: incorrectLrm.spinner
//   Created By: Gaurav Khandpur
//   Created on: 11/11/2003
//
//      Purpose: Incorrect Spinner Program
//
//  Description: Parser should give errors for each statement.  To test,
//                       make a copy and run it through the parser (front end).
//                       Keep deleting each statement as you proceed to test each
//                       invalid statement.
//
////////////////////////////////////////////////////////////////////////
/*
 * $Log: incorrectLrm.spinner,v $
 * Revision 1.2  2003/12/01 22:53:09  me133
 * o Minor comment change
 *
 * Revision 1.1  2003/12/01 06:30:26  me133
 * o Created
 *
 */

//testing comments

/* this is a comment
//this is also comment

/*this is a nested comment
*/

*/

//testing identifiers
```

```
//cannot start with digit
double 8indentifier;

//identifier cannot be keyword or boolean/null literal
double null;
double double;
double boolean;
double all;
double repeat;
double else;
double return;
double break;
double for;
double then;
double while;
double func;
double void;
double do;
double if;
double not;
double and;
double or;
double until;
double true;
double false;

//testing string ??

//testing variable declaration
//each variable should be declared separately
double a,b;
double c d;

//duplicate declaration, this will be error at compile time
//since they are in same scope

double x;
double x;

//testing variable declaration, it should be type followed by variable name followed by
//optional assignment.

name double;
name double := 5;
```

```
//testing assignment, it should be :=
double var = 4;

//testing import, it should be the first line of the SPINNER

import ("abc.spinner");

//local spoce has import, this is also error

{
import ("disclib.spinner");
}

//testing statements

//if statment
//missing "then"
if(2<3)
{
}
else callFunc();

//testing if statements use ; to end
//missing ;

{
break
}

//while..do
//it should be a check for expression instead of statement.
//should use = inplace of :=
while(c:=false) do{;}

//for statement
//missing init, expr and increment
for(;;;){}

//assignment inplace of expr
for(double i;i:=5;i++){}

//the 3rd part should be assignment
for(double a;a<=b;a>c){}

//testing return, i think the back deals with this
void callFunc1(){
```

```
a:=5;b:=6;
return a;
}

//testing concurrency operator
//block with standalone func. not allowed
{
a:=7;
} || callFunc2();

callFunc1();
||
{;}

//testing array initialization
double arr1[2]:={1,2};
```

E.2 Interpreter

E.2.1 test_all_constructs.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test_all_constructs.spinner,v 1.15 2003/12/17 1
//
//    File name: test_all_constructs.spinner
//   Created By: Marc Eaddy
//   Created on: 11/23/2003
//
//      Purpose: Simple test of every Spinner programming construct
//
//  Description:
//
//////////////////////////////////////////////////////////////////////
/**
 * $Log: test_all_constructs.spinner,v $
 * Revision 1.15  2003/12/17 14:09:44  me133
 * o Added in the time comments
 * o Changed the prefix for all spin script cmds to "1"
 * o Added test for counting backwards in a for statement
 *
 * Revision 1.14  2003/12/02 05:33:10  me133
 * o Added trivial waitUntil test
 *
 * Revision 1.13  2003/12/02 02:17:56  me133
 * o Added variable and array test
 *
 * Revision 1.12  2003/12/01 22:51:50  me133
 * o Fixed comments
 *
 * Revision 1.5  2003/12/01 22:36:26  me133
 * o Per William's request, added readfile command
 * o Got rid of extra newlines
 *
 * Revision 1.4  2003/11/30 20:12:36  me133
 * o For statement now uses a variable
 * o Function example now uses parameters and returns a Double
 *
 * Revision 1.3  2003/11/24 05:45:02  me133
 * o Comments
 *
 * Revision 1.2  2003/11/24 05:19:15  me133
 * o Now tests function calling
 *
 * Revision 1.1  2003/11/24 03:40:10  me133
 * o Created
 */
                                // 1, --------------- Time: 0;
// Initialization
createDiscs(6);

setFile(all, "Aeolean-Harp");   // 1, all-readfile Aeolean-Harp;

// Builtin function call and Addition
setGain(1, 1+2);  // 1, 1-gain 3.0;

// Subtraction (positive result)
setGain(1, 6-2);  // 1, 1-gain 4.0;

// Subtraction (negative result)
setGain(1, 3-19);  // 1, 1-gain -16.0;

// Multiplication
setGain(1, 2*3);  // 1, 1-gain 6.0;
```

```
// Division
setGain(1, 10/5);  // 1, 1-gain 2.0;

// Modulo
setGain(1, 6%5);  // 1, 1-gain 1.0;

// Negative numbers
setGain(1, -2 * 3);  // 1, 1-gain -6.0;

// Associativity
setGain(1, 1 - 3 + 4);  // 1, 1-gain 2.0;

// Precedence
setGain(1, 1 - 3 * 2 + 4);  // 1, 1-gain -1.0;

// if-then and Equality
if (1 = 1) then {
setGain(2, 1); // 1, 2-gain 1.0;
}

// if-then and Equality and "and"
if (1 = 1 and 2 = 2) then {
setGain(2, 2); // 1, 2-gain 2.0;
}

// if-then and Equality and "or"
if (1 = 2 or 2 = 2) then {
setGain(2, 3); // 1, 2-gain 3.0;
}

// if-then-else and Equality
if (1 = 2) then {
setGain(2, 0.1);
}
else {
setGain(2, 4); // 1, 2-gain 4.0;
}

// if-then-else-if-then and Equality
if (1 = 2) then {
setGain(2, 0.2);
}
else if (1 = 1) then {
setGain(2, 5); // 1, 2-gain 5.0;
}
```

```
// if-then-else-if-then-else and Equality
if (1 = 2) then {
setGain(2, 0.3);
}
else if (1 = 3) then {
setGain(2, 0.4);
}
else {
setGain(2, 6); // 1, 2-gain 6.0;
}

// if-then and Inequality
if (1 != 2) then {
setGain(3, 1); // 1, 3-gain 1.0;
}
else {
setGain(3, 0.1);
}

// if-then and Equality and "and"
if (1 != 2 and 2 != 3) then {
setGain(3, 2); // 1, 3-gain 2.0;
}
else {
setGain(3, 0.2);
}

// if-then and Equality and "or"
if (1 != 1 or 2 != 3) then {
setGain(3, 3); // 1, 3-gain 3.0;
}
else {
setGain(3, 0.3);
}

// LT
if (1 < 2) then {
setGain(4, 1); // 1, 4-gain 1.0;
}
else {
setGain(4, 0.1);
}

// LE
if (1 <= 1) then {
setGain(4, 2); // 1, 4-gain 2.0;
```

```
}
else {
setGain(4, 0.2);
}

if (1 <= 2) then {
setGain(4, 3); // 1, 4-gain 3.0;
}
else {
setGain(4, 0.3);
}

// GT
if (2 > 1) then {
setGain(5, 1); // 1, 5-gain 1.0;
}
else {
setGain(5, 0.1);
}

// GE
if (1 >= 1) then {
setGain(5, 2); // 1, 5-gain 2.0;
}
else {
setGain(5, 0.2);
}

if (2 >= 1) then {
setGain(5, 3); // 1, 5-gain 3.0;
}
else {
setGain(5, 0.3);
}

// Not
if (not 1 = 2) then {
setGain(6, 1); // 1, 6-gain 1.0;
}
else {
setGain(6, 0);
}

// while // 1, 1-gain 0.0;
while (getTime() < 3) { // 1, --wait-- 1;
setGain(1, getTime()); // 1, --------------- Time: 1;
```

```
wait(1); // 1, 1-gain 1.0;
} // 1, --wait-- 1;
                                // 1, --------------- Time: 2;
                                // 1, 1-gain 2.0;
                                // 1, --wait-- 1;
                                // 1, --------------- Time: 3;

// for
for(double b := 2; getTime() < 6; wait(1)) {// 1, 2-gain 3.0;
setGain(b, getTime()); // 1, --wait-- 1;
}                               // 1, --------------- Time: 4;
// 1, 2-gain 4.0;
// 1, --wait-- 1;
                                // 1, --------------- Time: 5;
// 1, 2-gain 5.0;
// 1, --wait-- 1;
                                // 1, --------------- Time: 6;

// repeat-until // 1, 3-gain 6.0;
repeat { // 1, --wait-- 1;
setGain(3, getTime()); // 1, --------------- Time: 7;
wait(1); // 1, 3-gain 7.0;
} // 1, --wait-- 1;
until (getTime() >= 9); // 1, --------------- Time: 8;
                                // 1, 3-gain 8.0;
                                // 1, --wait-- 1;
                                // 1, --------------- Time: 9;

// Function definition
double func testFunc(double a, double b) {
setGain(a, getTime());
wait(b);
return 5;
        //return 0; // Should be ignored!
}

//testFunc(4, 1); // 1, 4-gain 9.0;
// 1, --wait-- 1;
testFunc(testFunc(4, 1), 1); // 1, --------------- Time: 10;
                                // 1, 5-gain 10.0;
// 1, --wait-- 1;
                                // 1, --------------- Time: 11;

// Variable decl with initial value
double nCount := 1.0;
setGain(1, nCount);             // 1, 1-gain 1.0;
```

```
nCount := 2;
setGain(1, nCount);             // 1, 1-gain 2.0;

// Variable decl without initial value
double nCount2;
setGain(2, nCount2);            // 1, 2-gain 0.0; (Should be compile error!)
nCount2 := 3;
setGain(2, nCount2);            // 1, 2-gain 3.0;

// Array decl
double ary[5];

// Initialize array using for-loop
for(double i := 1; i <= 5; i++)
{
    ary[i] := i;
}

for(double i := 1; i <= 5; i++) // 1, 1-gain 1.0;
{                               // 1, 2-gain 2.0;
    setGain(ary[i], ary[i]);    // 1, 3-gain 3.0;
}                               // 1, 4-gain 4.0;
                                // 1, 5-gain 5.0;
// Trivial test for waitUntil
waitUntil(true);

wait(1);                        // 1, --wait-- 1;
                                // 1, -------------- Time: 12;

// for
for(double c := 6; c >= 1; --c) {// 1, 6-gain 6.0;
setGain(c, c);          // 1, 5-gain 5.0;
} // 1, 4-gain 4.0;
                                // 1, 3-gain 3.0;
                                // 1, 2-gain 2.0;
                                // 1, 1-gain 1.0;
```

E.2.2 test1.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test1.spinner,v 1.2 2003/12/12 23:54:29 gk2017
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/2/2003
//
//      Purpose: testing Spinner programs that end with a comment, was giving parse error
//
//  Description:
//  Status: Doesnot work yet, Sangho?
//
///////////////////////////////////////////////////////////////////////

createDiscs(5);
setFile(1,"Aeolean-Harp");  //1,1-readFile Aeolean-Harp;
seek(all,5,5000);  //1,all-line 5.0 5000;
wait(5000);  //1,wait 5000;
seek(all,0,2500);  //1,all-line 0.0 2500;
//wait(2000);  //1,wait 2000;
//if(getPosition(1)=0) then {
//   seek(1, 2, 2000);
//} else {
//   waitUntil(getPosition(1)=0,-1);
//}

//wait(70);
;
//asokfja
```

E.2.3 test2.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test2.spinner,v 1.3 2003/12/12 23:54:29 gk2017
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/2/2003
//
//      Purpose: testing Spinner(general)
//
//  Description:
//  Status: Not working: waitUntil, strange output :  1, 1-line 4.5 0, getPosition value is 9.0
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: test2.spinner,v $
 * Revision 1.3  2003/12/12 23:54:29  gk2017
 * *** empty log message ***
 *
 * Revision 1.1  2003/12/03 03:25:04  me133
 * o Created
 *
 */

createDiscs(5);
setFile(1,"Aeolean-Harp");  //1,1-readFile Aeolean-Harp; pass

if(isFileSet(1)) then {
setFile(2,"10-beeps");  //1,2-readFile 10-beeps; pass
setSpinMult(2,-0.5);    //1,2-spinmult -0.5; pass
}

//some variable declarations
double a:=1.0;
double b:=getDiscsSize();

//a function definition
double func callFunc1(double var1,double var2) {
return var2-var1-1;
}

if(getSpinMult(2)= -.5) then {
setFile(callFunc1(a,b),"p-phase-1");  //1,3-readFile p-phase-1; pass
}

if(isFileSet(4)) then {
setFile(3,"Ants");     //doesnot output spin for this , pass
}

if(getPosition(4)=0) then { //disc 4 is at 0.
seek(4,2,2000);   //1,4-line 2.0 2000; pass but also outputs 1,4-speed 0.0 0.0
wait(5000);   //1,--wait-- 5000; pass  now disc 4 is at 2.
seek(4,getPosition(4),5000);   //1,4-line 2.0 5000;  pass but speed 0.0 0.0 again
}

seek(3,4,4000);     //1,3-line 4.0 4000; pass but speed 0.0 0.0 again as above
seek(3,3,3000);   //1,3-line 3.0 3000; pass

//CHECKING setSpeed
if(getPosition(2)=0) then {
setSpeed(2,1.0,1000);  //1,2-speed 1.0 1000; pass
wait(3000);      //1,--wait-- 3000; pass
}
seek(1,getPosition(2),2000);  //1,1-line 3.0 2000; // ouput has 9 as position, pass?
wait(1000);        //1,--wait-- 1000; pass
seek(1,getPosition(2),2000);  //1,1-line 4.0 2000; pass?

//strange output ??:  1, 1-line 4.5 0; William do you know why is this?

//CHECKING getSpeed
setSpeed(1,getSpeed(2),1000);  //1,1-speed 1.0 1000;  pass

setSpeed(1,1.0,1000);  //1,1-speed 1.0 1000; pass
wait(2000); //1,--wait-- 2000; pass
setSpeed(3,getSpeed(1),1000); //1,3-speed 1.0 1000; pass

waitUntil(getTime()<=13000,1000); //ERROR  programs keeps evaluating and then throws
//java.lang.OutOfMemoryError

//ERROR-Prgram keeps evaluating this statement
//waitUntil(getTime()<=13000);
```

E.2.4 test3.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test3.spinner,v 1.2 2003/12/12 23:54:29 gk2017
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/2/2003
//
//      Purpose: //CHECKING DYNAMIC OR STATIC SCOPING
//
//  Description:
//
//////////////////////////////////////////////////////////////////////



double b:=5;

double func callfunc1() {

//b:=4;
return b;
}

createDiscs(4);
double a:=1;
if(true) then {
double b:=10;
double c:=callfunc1();
   seek(1,c,1000);  //1,1-line 10 1000;
}
```

E.2.5 test4.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test4.spinner,v 1.2 2003/12/12 23:54:29 gk2017
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/2/2003
//
//      Purpose: Testing Symbol table addition
//
//  Description:
//
//////////////////////////////////////////////////////////////////////

double a:=4;
double b:=5;
createDiscs(4);

//if(a<b) then {
//a:=10;
//setGain(1,1);
//}
//setGain(2,a);
//a=b;

{
setGain(3,1.0);
//wait(3000);
setGain(2,2.0);
}||
{
setGain(1,1.0);
//wait(2000);
setGain(4,1.0);
}
```

E.2.6 test6.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test6.spinner,v 1.1 2003/12/12 23:54:29 gk2017
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/12/2003
//
//      Purpose: testing Spinner(general)
//
//  Description: includes test for '--' and undeclared variable inside for
//  Status: -- is not handled properly inside for, undeclared variable inside for error keeps looping
//
//////////////////////////////////////////////////////////////////////

//test for with '--'
for(double i:=6;i>=1;i--) {}

//undeclared variable i inside for
for(i:=6;i<=6;i++) {}
```

E.2.7 parallel1.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/parallel1.spinner,v 1.4 2003/12/01 22:53:09 me1
//
//    File name: parallel1.spinner
//   Created By: Marc Eaddy
//   Created on: 11/30/2003
//
//      Purpose: Spinner program that excercizes parallels statements
//
//  Description:
//
```

```
////////////////////////////////////////////////////////////////////////
/*
 * $Log: parallel1.spinner,v $
 * Revision 1.4  2003/12/01 22:53:09  me133
 * o Minor comment change
 *
 * Revision 1.3  2003/12/01 22:36:05  me133
 * o Per William's request, added readfile command
 *
 * Revision 1.2  2003/12/01 06:55:02  me133
 * o Added scoping test for variables in parallel branches
 *
 * Revision 1.1  2003/12/01 06:30:26  me133
 * o Created
 *
 */

createDiscs(6);

setFile(all, "Aeolean-Harp");

wait(10);

{
    double a := 1;
    setGain(a, 1.0);
    setGain(a, 1.1);
    wait(10);
    setGain(a, 1.2);
} || {
    double a := 2;
    setGain(a, 2.0);
    setGain(a, 2.1);
    wait(20);
    setGain(a, 2.2);
}

setGain(3, 3.0);
```

E.2.8 parallel2.spinner

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/parallel2.spinner,v 1.5 2003/12/17 14:07:11 me1
```

```
//
//    File name: parallel2.spinner
//   Created By: Marc Eaddy
//   Created on: 11/30/2003
//
//      Purpose: Spinner program that excercizes parallels statements
//
//  Description: This is the sample from the LRM with a few bug
//                      fixes and some other minor changes.
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: parallel2.spinner,v $
 * Revision 1.5  2003/12/17 14:07:11  me133
 * o Created real sin and cosine
 * o Modified algorithm slightly so that it would terminate
 *
 * Revision 1.4  2003/12/01 22:53:09  me133
 * o Minor comment change
 *
 * Revision 1.3  2003/12/01 22:36:05  me133
 * o Per William's request, added readfile command
 *
 * Revision 1.2  2003/12/01 06:31:24  me133
 * o Changed comment
 *
 * Revision 1.1  2003/12/01 06:30:26  me133
 * o Created
 */

createDiscs(5);

setFile(all, "Aeolean-Harp");

double func sin(double degrees) {

    double sin_table[360];

    sin_table[1] := 0;
    sin_table[2] := 0.841470984807897;
    sin_table[3] := 0.909297426825682;
    sin_table[4] := 0.141120008059867;
    sin_table[5] := -0.756802495307928;
    sin_table[6] := -0.958924274663138;
    sin_table[7] := -0.279415498198926;
    sin_table[8] := 0.656986598718789;
```

```
sin_table[9] := 0.989358246623382;
sin_table[10] := 0.412118485241757;
sin_table[11] := -0.54402111088937;
sin_table[12] := -0.999990206550703;
sin_table[13] := -0.536572918000435;
sin_table[14] := 0.420167036826641;
sin_table[15] := 0.99060735569487;
sin_table[16] := 0.650287840157117;
sin_table[17] := -0.287903316665065;
sin_table[18] := -0.961397491879557;
sin_table[19] := -0.750987246771676;
sin_table[20] := 0.149877209662952;
sin_table[21] := 0.912945250727628;
sin_table[22] := 0.836655638536056;
sin_table[23] := -0.00885130929040388;
sin_table[24] := -0.846220404175171;
sin_table[25] := -0.905578362006624;
sin_table[26] := -0.132351750097773;
sin_table[27] := 0.762558450479603;
sin_table[28] := 0.956375928404503;
sin_table[29] := 0.270905788307869;
sin_table[30] := -0.663633884212968;
sin_table[31] := -0.988031624092862;
sin_table[32] := -0.404037645323065;
sin_table[33] := 0.551426681241691;
sin_table[34] := 0.999911860107267;
sin_table[35] := 0.529082686120024;
sin_table[36] := -0.428182669496151;
sin_table[37] := -0.991778853443116;
sin_table[38] := -0.643538133357;
sin_table[39] := 0.296368578709385;
sin_table[40] := 0.963795386284088;
sin_table[41] := 0.745113160479349;
sin_table[42] := -0.158622668804709;
sin_table[43] := -0.916521547915634;
sin_table[44] := -0.831774742628598;
sin_table[45] := 0.0177019251054136;
sin_table[46] := 0.850903524534118;
sin_table[47] := 0.901788347648809;
sin_table[48] := 0.123573122745224;
sin_table[49] := -0.768254661323667;
sin_table[50] := -0.953752652759472;
sin_table[51] := -0.262374853703929;
sin_table[52] := 0.670229175843375;
sin_table[53] := 0.986627592040485;
sin_table[54] := 0.395925150181834;
```

```
sin_table[55] := -0.558789048851616;
sin_table[56] := -0.99975517335862;
sin_table[57] := -0.521551002086912;
sin_table[58] := 0.436164755247825;
sin_table[59] := 0.992872648084537;
sin_table[60] := 0.636738007139138;
sin_table[61] := -0.304810621102217;
sin_table[62] := -0.966117770008393;
sin_table[63] := -0.739180696649223;
sin_table[64] := 0.167355700302807;
sin_table[65] := 0.920026038196791;
sin_table[66] := 0.826828679490103;
sin_table[67] := -0.0265511540239668;
sin_table[68] := -0.855519978975322;
sin_table[69] := -0.897927680689291;
sin_table[70] := -0.114784813783187;
sin_table[71] := 0.773890681557889;
sin_table[72] := 0.951054653254375;
sin_table[73] := 0.253823362762036;
sin_table[74] := -0.676771956887308;
sin_table[75] := -0.985146260468247;
sin_table[76] := -0.38778163540943;
sin_table[77] := 0.56610763689818;
sin_table[78] := 0.999520158580731;
sin_table[79] := 0.513978455987535;
sin_table[80] := -0.444112668707508;
sin_table[81] := -0.993888653923375;
sin_table[82] := -0.629887994274454;
sin_table[83] := 0.313228782433085;
sin_table[84] := 0.968364461100185;
sin_table[85] := 0.733190320073292;
sin_table[86] := -0.176075619948587;
sin_table[87] := -0.92345844700406;
sin_table[88] := -0.821817836630823;
sin_table[89] := 0.0353983027336607;
sin_table[90] := 0.860069405812453;
sin_table[91] := 0.893996663600558;
sin_table[92] := 0.105987511751157;
sin_table[93] := -0.779466069615805;
sin_table[94] := -0.948282141269947;
sin_table[95] := -0.245251985467654;
sin_table[96] := 0.683261714736121;
sin_table[97] := 0.983587745434345;
sin_table[98] := 0.379607739027522;
sin_table[99] := -0.573381871990423;
sin_table[100] := -0.999206834186354;
```

```
sin_table[101] := -0.506365641109759;
sin_table[102] := 0.452025787178351;
sin_table[103] := 0.994826791358406;
sin_table[104] := 0.622988631442349;
sin_table[105] := -0.321622403162531;
sin_table[106] := -0.970535283537485;
sin_table[107] := -0.727142500080853;
sin_table[108] := 0.184781744560667;
sin_table[109] := 0.926818505417785;
sin_table[110] := 0.816742606636317;
sin_table[111] := -0.044242678085071;
sin_table[112] := -0.864551448610608;
sin_table[113] := -0.889995604366833;
sin_table[114] := -0.097181905893209;
sin_table[115] := 0.78498038868131;
sin_table[116] := 0.94543533402477;
sin_table[117] := 0.236661393364286;
sin_table[118] := -0.689697940935389;
sin_table[119] := -0.981952169044084;
sin_table[120] := -0.37140410143809;
sin_table[121] := 0.580611184212314;
sin_table[122] := 0.99881522472358;
sin_table[123] := 0.498713153896394;
sin_table[124] := -0.459903490689591;
sin_table[125] := -0.995686986889179;
sin_table[126] := -0.616040459188656;
sin_table[127] := 0.329990825673782;
sin_table[128] := 0.972630067242408;
sin_table[129] := 0.721037710501732;
sin_table[130] := -0.193473392038468;
sin_table[131] := -0.930105950186762;
sin_table[132] := -0.8116033871367;
sin_table[133] := 0.0530835871460582;
sin_table[134] := 0.868965756214236;
sin_table[135] := 0.885924816459948;
sin_table[136] := 0.0883686861040014;
sin_table[137] := -0.790433206722889;
sin_table[138] := -0.942514454558251;
sin_table[139] := -0.228052259500861;
sin_table[140] := 0.696080131224742;
sin_table[141] := 0.980239659440312;
sin_table[142] := 0.363171365373259;
sin_table[143] := -0.587795007167407;
sin_table[144] := -0.998345360873918;
sin_table[145] := -0.491021593898469;
sin_table[146] := 0.467745162045133;
```

```
sin_table[147] := 0.996469173121774;
sin_table[148] := 0.609044021883292;
sin_table[149] := -0.338333394324277;
sin_table[150] := -0.974648648094495;
sin_table[151] := -0.714876429629165;
sin_table[152] := 0.202149881415654;
sin_table[153] := 0.933320523748862;
sin_table[154] := 0.806400580775486;
sin_table[155] := -0.0619203372560573;
sin_table[156] := -0.873311982774648;
sin_table[157] := -0.881784618814781;
sin_table[158] := -0.0795485428747221;
sin_table[159] := 0.795824096527455;
sin_table[160] := 0.939519731713148;
sin_table[161] := 0.219425258379005;
sin_table[162] := -0.702407785577371;
sin_table[163] := -0.97845035079338;
sin_table[164] := -0.354910175844935;
sin_table[165] := 0.594932778023209;
sin_table[166] := 0.997797279449891;
sin_table[167] := 0.483291563728257;
sin_table[168] := -0.475550186871899;
sin_table[169] := -0.99717328877408;
sin_table[170] := -0.601999867677605;
sin_table[171] := 0.34664945549703;
sin_table[172] := 0.976590867943566;
sin_table[173] := 0.708659140182323;
sin_table[174] := -0.210810532913481;
sin_table[175] := -0.936461974251213;
sin_table[176] := -0.801134595178041;
sin_table[177] := 0.0707522360803452;
sin_table[178] := 0.877589787777116;
sin_table[179] := 0.877575335804269;
sin_table[180] := 0.0707221672389912;
sin_table[181] := -0.80115263573383;
sin_table[182] := -0.936451400117644;
sin_table[183] := -0.210781065900192;
sin_table[184] := 0.708680408239208;
sin_table[185] := 0.976584383290629;
sin_table[186] := 0.346621180094276;
sin_table[187] := -0.602023937555283;
sin_table[188] := -0.997171023392149;
sin_table[189] := -0.475523669012058;
sin_table[190] := 0.483317953667963;
sin_table[191] := 0.9977992786806;
sin_table[192] := 0.594908548461427;
```

```
sin_table[193] := -0.354938357651846;
sin_table[194] := -0.978456574622113;
sin_table[195] := -0.702386329268492;
sin_table[196] := 0.219454667994064;
sin_table[197] := 0.939530055569931;
sin_table[198] := 0.795805842919647;
sin_table[199] := -0.0795785916642835;
sin_table[200] := -0.88179883606755;
sin_table[201] := -0.873297297213995;
sin_table[202] := -0.0618902507187207;
sin_table[203] := 0.80641840686583;
sin_table[204] := 0.93330970016696;
sin_table[205] := 0.202120359312791;
sin_table[206] := -0.714897507767764;
sin_table[207] := -0.97464190312541;
sin_table[208] := -0.338305027540978;
sin_table[209] := 0.60906793019106;
sin_table[210] := 0.996466641766108;
sin_table[211] := 0.467718518342759;
sin_table[212] := -0.491047853850463;
sin_table[213] := -0.998347093796772;
sin_table[214] := -0.587770619819841;
sin_table[215] := 0.363199451376361;
sin_table[216] := 0.980245621957223;
sin_table[217] := 0.696058488344911;
sin_table[218] := -0.228081609413528;
sin_table[219] := -0.942524527329403;
sin_table[220] := -0.790414741493181;
sin_table[221] := 0.0883987124875315;
sin_table[222] := 0.885938797878757;
sin_table[223] := 0.868950838216349;
sin_table[224] := 0.0530534852699353;
sin_table[225] := -0.811620997364974;
sin_table[226] := -0.930094878004525;
sin_table[227] := -0.193443817159008;
sin_table[228] := 0.721058597070632;
sin_table[229] := 0.972623062485624;
sin_table[230] := 0.329962369732397;
sin_table[231] := -0.616064204053364;
sin_table[232] := -0.995684189758103;
sin_table[233] := -0.459876723232143;
sin_table[234] := 0.498739281803281;
sin_table[235] := 0.998816691202808;
sin_table[236] := 0.580586640989645;
sin_table[237] := -0.371432089436923;
sin_table[238] := -0.981957869782025;
```

```
sin_table[239] := -0.689676113180267;
sin_table[240] := 0.236690681275077;
sin_table[241] := 0.945445154921117;
sin_table[242] := 0.784961713276403;
sin_table[243] := -0.0972119075182243;
sin_table[244] := -0.890009348856277;
sin_table[245] := -0.864536299344272;
sin_table[246] := -0.0442125632285597;
sin_table[247] := 0.816759999622809;
sin_table[248] := 0.926807185502688;
sin_table[249] := 0.184752119221718;
sin_table[250] := -0.727163193443649;
sin_table[251] := -0.970528019541805;
sin_table[252] := -0.321593860292504;
sin_table[253] := 0.623012211003653;
sin_table[254] := 0.994823728671067;
sin_table[255] := 0.451998898062983;
sin_table[256] := -0.506391634924491;
sin_table[257] := -0.999208034107063;
sin_table[258] := -0.573357174815543;
sin_table[259] := 0.379635626829303;
sin_table[260] := 0.983593183946681;
sin_table[261] := 0.683239703815851;
sin_table[262] := -0.245281209081943;
sin_table[263] := -0.948291709522049;
sin_table[264] := -0.779447185498863;
sin_table[265] := 0.106017486267114;
sin_table[266] := 0.894010170083794;
sin_table[267] := 0.86005402646457;
sin_table[268] := 0.035368177256176;
sin_table[269] := -0.82183501101284;
sin_table[270] := -0.923446880242987;
sin_table[271] := -0.176045946471211;
sin_table[272] := 0.733210818608718;
sin_table[273] := 0.968356938434724;
sin_table[274] := 0.31320015487067;
sin_table[275] := -0.629911406684961;
sin_table[276] := -0.993885325919726;
sin_table[277] := -0.44408566004091;
sin_table[278] := 0.514004313673569;
sin_table[279] := 0.99952109184891;
sin_table[280] := 0.566082787706044;
sin_table[281] := -0.387809420829229;
sin_table[282] := -0.985151436328885;
sin_table[283] := -0.676749764526383;
sin_table[284] := 0.253852519790234;
```

```
sin_table[285] := 0.951063968112585;
sin_table[286] := 0.773871590208432;
sin_table[287] := -0.114814758841666;
sin_table[288] := -0.897940948108125;
sin_table[289] := -0.855504370750821;
sin_table[290] := -0.026521020285756;
sin_table[291] := 0.826845633922081;
sin_table[292] := 0.920014225495965;
sin_table[293] := 0.167325981011839;
sin_table[294] := -0.739200998751274;
sin_table[295] := -0.96610998926253;
sin_table[296] := -0.304781911090303;
sin_table[297] := 0.636761250564552;
sin_table[298] := 0.992869055025318;
sin_table[299] := 0.436137629146049;
sin_table[300] := -0.52157672161837;
sin_table[301] := -0.999755839901149;
sin_table[302] := -0.558764049589089;
sin_table[303] := 0.395952831042741;
sin_table[304] := 0.986632504843911;
sin_table[305] := 0.670206803780506;
sin_table[306] := -0.262403941861664;
sin_table[307] := -0.953761713493999;
sin_table[308] := -0.768235364237447;
sin_table[309] := 0.123603036000113;
sin_table[310] := 0.901801374963775;
sin_table[311] := 0.85088768865586;
sin_table[312] := 0.0176717854673709;
sin_table[313] := -0.831791475782204;
sin_table[314] := -0.916509490200547;
sin_table[315] := -0.158592906028573;
sin_table[316] := 0.745133264557413;
sin_table[317] := 0.963787348067422;
sin_table[318] := 0.296339788497322;
sin_table[319] := -0.643561205976262;
sin_table[320] := -0.991774995609833;
sin_table[321] := -0.428155428084452;
sin_table[322] := 0.529108265481853;
sin_table[323] := 0.999912259871926;
sin_table[324] := 0.551401533867395;
sin_table[325] := -0.404065219456361;
sin_table[326] := -0.98803627345417;
sin_table[327] := -0.663611334200943;
sin_table[328] := 0.270934805316165;
sin_table[329] := 0.956384734305463;
sin_table[330] := 0.762538949168494;
```

```
    sin_table[331] := -0.132381629205452;
    sin_table[332] := -0.905591148197067;
    sin_table[333] := -0.846204341883851;
    sin_table[334] := -0.00882116611388588;
    sin_table[335] := 0.836672149100295;
    sin_table[336] := 0.912932948942968;
    sin_table[337] := 0.149847405733478;
    sin_table[338] := -0.751007151250654;
    sin_table[339] := -0.961389196821861;
    sin_table[340] := -0.287874448508486;
    sin_table[341] := 0.650310740162553;
    sin_table[342] := 0.990603233389774;
    sin_table[343] := 0.420139682239307;
    sin_table[344] := -0.536598355188564;
    sin_table[345] := -0.999990339506171;
    sin_table[346] := -0.543995817373532;
    sin_table[347] := 0.412145950487085;
    sin_table[348] := 0.989362632178309;
    sin_table[349] := 0.65696387252434;
    sin_table[350] := -0.279444441784382;
    sin_table[351] := -0.958932825040613;
    sin_table[352] := -0.756782791299803;
    sin_table[353] := 0.141149850679391;
    sin_table[354] := 0.909309970889841;
    sin_table[355] := 0.841454697361953;
    sin_table[356] := 0;
    sin_table[357] := -0.841487271489211;
    sin_table[358] := -0.90928488193526;
    sin_table[359] := -0.14109016531211;
    sin_table[360] := 0.75682219862836;

    return sin_table[degrees];
}

double func cos(double degrees) {

    double cos_table[360];

    cos_table[1] := 1;
    cos_table[2] := 0.54030230586814;
    cos_table[3] := -0.416146836547142;
    cos_table[4] := -0.989992496600445;
    cos_table[5] := -0.653643620863612;
    cos_table[6] := 0.283662185463226;
    cos_table[7] := 0.960170286650366;
    cos_table[8] := 0.753902254343305;
```

```
cos_table[9] := -0.145500033808614;
cos_table[10] := -0.911130261884677;
cos_table[11] := -0.839071529076452;
cos_table[12] := 0.00442569798805079;
cos_table[13] := 0.843853958732492;
cos_table[14] := 0.907446781450196;
cos_table[15] := 0.136737218207834;
cos_table[16] := -0.759687912858821;
cos_table[17] := -0.957659480323385;
cos_table[18] := -0.275163338051597;
cos_table[19] := 0.66031670824408;
cos_table[20] := 0.988704618186669;
cos_table[21] := 0.408082061813392;
cos_table[22] := -0.547729260224268;
cos_table[23] := -0.999960826394637;
cos_table[24] := -0.532833020333398;
cos_table[25] := 0.424179007336997;
cos_table[26] := 0.991202811863474;
cos_table[27] := 0.64691932232864;
cos_table[28] := -0.292138808733836;
cos_table[29] := -0.962605866313567;
cos_table[30] := -0.748057529689;
cos_table[31] := 0.154251449887584;
cos_table[32] := 0.914742357804531;
cos_table[33] := 0.83422336050651;
cos_table[34] := -0.0132767472230595;
cos_table[35] := -0.848570274784605;
cos_table[36] := -0.903692205091507;
cos_table[37] := -0.127963689627405;
cos_table[38] := 0.765414051945343;
cos_table[39] := 0.955073644047295;
cos_table[40] := 0.266642932359937;
cos_table[41] := -0.666938061652262;
cos_table[42] := -0.987339277523826;
cos_table[43] := -0.399985314988351;
cos_table[44] := 0.555113301520626;
cos_table[45] := 0.999843308647691;
cos_table[46] := 0.52532198881773;
cos_table[47] := -0.432177944884778;
cos_table[48] := -0.992335469150929;
cos_table[49] := -0.6401443394692;
cos_table[50] := 0.300592543743637;
cos_table[51] := 0.964966028492113;
cos_table[52] := 0.742154196813783;
cos_table[53] := -0.162990780795705;
cos_table[54] := -0.918282786212119;
```

```
cos_table[55] := -0.82930983286315;
cos_table[56] := 0.0221267562619557;
cos_table[57] := 0.853220107722584;
cos_table[58] := 0.899866826969194;
cos_table[59] := 0.119180135448819;
cos_table[60] := -0.771080222975845;
cos_table[61] := -0.952412980415156;
cos_table[62] := -0.258101635938267;
cos_table[63] := 0.673507162323586;
cos_table[64] := 0.98589658158255;
cos_table[65] := 0.39185723042955;
cos_table[66] := -0.562453851238172;
cos_table[67] := -0.99964745596635;
cos_table[68] := -0.517769799789505;
cos_table[69] := 0.440143022496041;
cos_table[70] := 0.993390379722272;
cos_table[71] := 0.6333192030863;
cos_table[72] := -0.309022728166071;
cos_table[73] := -0.967250588273882;
cos_table[74] := -0.736192718227316;
cos_table[75] := 0.171717341830778;
cos_table[76] := 0.921751269724749;
cos_table[77] := 0.824331331107558;
cos_table[78] := -0.0309750317312165;
cos_table[79] := -0.857803093244988;
cos_table[80] := -0.895970946790963;
cos_table[81] := -0.110387243839048;
cos_table[82] := 0.776685982021631;
cos_table[83] := 0.949677697882543;
cos_table[84] := 0.249540117973338;
cos_table[85] := -0.680023495587339;
cos_table[86] := -0.984376643394042;
cos_table[87] := -0.383698444949742;
cos_table[88] := 0.569750334265312;
cos_table[89] := 0.999373283695125;
cos_table[90] := 0.510177044941669;
cos_table[91] := -0.44807361612917;
cos_table[92] := -0.994367460928202;
cos_table[93] := -0.626444447910339;
cos_table[94] := 0.317428701519702;
cos_table[95] := 0.969459366669988;
cos_table[96] := 0.73017356099482;
cos_table[97] := -0.180430449291084;
cos_table[98] := -0.925147536596414;
cos_table[99] := -0.819288245291459;
cos_table[100] := 0.0398208803931389;
```

```
cos_table[101] := 0.862318872287684;
cos_table[102] := 0.89200486978816;
cos_table[103] := 0.101585703696621;
cos_table[104] := -0.782230889887116;
cos_table[105] := -0.946868010751213;
cos_table[106] := -0.240959049236201;
cos_table[107] := 0.686486550906984;
cos_table[108] := 0.982779582041221;
cos_table[109] := 0.375509597767012;
cos_table[110] := -0.577002178942952;
cos_table[111] := -0.999020813314648;
cos_table[112] := -0.502544319145385;
cos_table[113] := 0.455969104444276;
cos_table[114] := 0.995266636217131;
cos_table[115] := 0.61952061255921;
cos_table[116] := -0.325809805219964;
cos_table[117] := -0.971592190628802;
cos_table[118] := -0.724097196700474;
cos_table[119] := 0.189129420528958;
cos_table[120] := 0.928471320739076;
cos_table[121] := 0.814180970526562;
cos_table[122] := -0.0486636092001539;
cos_table[123] := -0.86676709105198;
cos_table[124] := -0.887968906691855;
cos_table[125] := -0.0927762045976609;
cos_table[126] := 0.787714512144234;
cos_table[127] := 0.943984139152314;
cos_table[128] := 0.232359102029658;
cos_table[129] := -0.692895821920165;
cos_table[130] := -0.981105522649388;
cos_table[131] := -0.367291330454696;
cos_table[132] := 0.584208817109289;
cos_table[133] := 0.998590072439991;
cos_table[134] := 0.49487222040343;
cos_table[135] := -0.463828868851872;
cos_table[136] := -0.996087835141185;
cos_table[137] := -0.6125482394961;
cos_table[138] := 0.334165382630761;
cos_table[139] := 0.973648893049518;
cos_table[140] := 0.717964101410472;
cos_table[141] := -0.197813574004268;
cos_table[142] := -0.93172236174352;
cos_table[143] := -0.809009906953598;
cos_table[144] := 0.0575025253491242;
cos_table[145] := 0.871147401032343;
cos_table[146] := 0.8838633737085;
```

```
cos_table[147] := 0.0839594367418485;
cos_table[148] := -0.793136419166478;
cos_table[149] := -0.941026309029144;
cos_table[150] := -0.223740950135584;
cos_table[151] := 0.699250806478375;
cos_table[152] := 0.979354596376429;
cos_table[153] := 0.359044286891116;
cos_table[154] := -0.591369684144325;
cos_table[155] := -0.9980810948185;
cos_table[156] := -0.487161349803341;
cos_table[157] := 0.471652293561339;
cos_table[158] := 0.996830993361718;
cos_table[159] := 0.60552787498699;
cos_table[160] := -0.342494779115907;
cos_table[161] := -0.975629312795237;
cos_table[162] := -0.711774755635724;
cos_table[163] := 0.206482229337811;
cos_table[164] := 0.93490040489975;
cos_table[165] := 0.803775459710974;
cos_table[166] := -0.0663369363356237;
cos_table[167] := -0.875459459043705;
cos_table[168] := -0.879688592495152;
cos_table[169] := -0.0751360908983532;
cos_table[170] := 0.798496186162556;
cos_table[171] := 0.937994752119442;
cos_table[172] := 0.215105268762141;
cos_table[173] := -0.7055510066863;
cos_table[174] := -0.977526940402531;
cos_table[175] := -0.350769113209131;
cos_table[176] := 0.5984842190141;
cos_table[177] := 0.997493920327152;
cos_table[178] := 0.479412311470322;
cos_table[179] := -0.479438765629173;
cos_table[180] := -0.997496052654355;
cos_table[181] := -0.598460069057858;
cos_table[182] := 0.350797342090421;
cos_table[183] := 0.977533294705597;
cos_table[184] := 0.705529644294206;
cos_table[185] := -0.215134707364621;
cos_table[186] := -0.93800520121695;
cos_table[187] := -0.798478038903032;
cos_table[188] := 0.0751661500081933;
cos_table[189] := 0.879702927248347;
cos_table[190] := 0.875444890134275;
cos_table[191] := 0.0663068583517113;
cos_table[192] := -0.803793393209672;
```

```
cos_table[193] := -0.934889705937235;
cos_table[194] := -0.206452734490879;
cos_table[195] := 0.711795928940826;
cos_table[196] := 0.975622697919444;
cos_table[197] := 0.342466457745517;
cos_table[198] := -0.605551864314651;
cos_table[199] := -0.996828594969431;
cos_table[200] := -0.471625712519911;
cos_table[201] := 0.487187675007006;
cos_table[202] := 0.998082960913557;
cos_table[203] := 0.591345375451585;
cos_table[204] := -0.359072421071653;
cos_table[205] := -0.979360689608925;
cos_table[206] := -0.699229256672974;
cos_table[207] := 0.223770330187178;
cos_table[208] := 0.941036507442989;
cos_table[209] := 0.793118059567917;
cos_table[210] := -0.0839894746225687;
cos_table[211] := -0.883877473182372;
cos_table[212] := -0.871132599108112;
cos_table[213] := -0.0574724308476655;
cos_table[214] := 0.80902762528643;
cos_table[215] := 0.931711413754233;
cos_table[216] := 0.197784025223722;
cos_table[217] := -0.717985083969714;
cos_table[218] := -0.973642018119255;
cos_table[219] := -0.334136970990171;
cos_table[220] := 0.612572066315684;
cos_table[221] := 0.996085170871722;
cos_table[222] := 0.463802163010418;
cos_table[223] := -0.494898414589402;
cos_table[224] := -0.998591672156699;
cos_table[225] := -0.58418435158457;
cos_table[226] := 0.367319367730245;
cos_table[227] := 0.981111354333927;
cos_table[228] := 0.692874086389823;
cos_table[229] := -0.232388421228523;
cos_table[230] := -0.943994086083478;
cos_table[231] := -0.787695941645058;
cos_table[232] := 0.0928062188958771;
cos_table[233] := 0.887982769781749;
cos_table[234] := 0.866752057272636;
cos_table[235] := 0.0486335005389691;
cos_table[236] := -0.814198472305347;
cos_table[237] := -0.928460124580761;
cos_table[238] := -0.189099820129863;
```

```
cos_table[239] := 0.72411798686993;
cos_table[240] := 0.9715850561827;
cos_table[241] := 0.325781305535148;
cos_table[242] := -0.619544275003953;
cos_table[243] := -0.995263706279229;
cos_table[244] := -0.455942275895124;
cos_table[245] := 0.502570380261423;
cos_table[246] := 0.999022146527674;
cos_table[247] := 0.576977558503058;
cos_table[248] := -0.37553753594093;
cos_table[249] := -0.982785151720906;
cos_table[250] := -0.68646463135462;
cos_table[251] := 0.240988305285259;
cos_table[252] := 0.946877705420381;
cos_table[253] := 0.782212109942271;
cos_table[254] := -0.101615692060797;
cos_table[255] := -0.892018495407942;
cos_table[256] := -0.862303607831082;
cos_table[257] := -0.0397907599311577;
cos_table[258] := 0.819305529144982;
cos_table[259] := 0.925136093146258;
cos_table[260] := 0.180400799592549;
cos_table[261] := -0.730194157145638;
cos_table[262] := -0.96945197326701;
cos_table[263] := -0.31740011602353;
cos_table[264] := 0.626467944126354;
cos_table[265] := 0.994364265551414;
cos_table[266] := 0.448046666974262;
cos_table[267] := -0.510202970945957;
cos_table[268] := -0.999374350300014;
cos_table[269] := -0.569725560839186;
cos_table[270] := 0.383726281833151;
cos_table[271] := 0.984381950632505;
cos_table[272] := 0.680001393730288;
cos_table[273] := -0.249569308580458;
cos_table[274] := -0.949687139530166;
cos_table[275] := -0.776666994102475;
cos_table[276] := 0.110417203919678;
cos_table[277] := 0.895984333873104;
cos_table[278] := 0.857787599307056;
cos_table[279] := 0.030944901828293;
cos_table[280] := -0.824348395681676;
cos_table[281] := -0.921739579879316;
cos_table[282] := -0.171687645155773;
cos_table[283] := 0.736213118745846;
cos_table[284] := 0.967242936493283;
```

```
cos_table[285] := 0.308994059098137;
cos_table[286] := -0.633342531232723;
cos_table[287] := -0.993386919156947;
cos_table[288] := -0.440115954846768;
cos_table[289] := 0.517795588650813;
cos_table[290] := 0.999648255879538;
cos_table[291] := 0.562428926766744;
cos_table[292] := -0.391884963841509;
cos_table[293] := -0.985901625963983;
cos_table[294] := -0.673484879893468;
cos_table[295] := 0.258130758816447;
cos_table[296] := 0.952422168301506;
cos_table[297] := 0.771061028570027;
cos_table[298] := -0.119210064898616;
cos_table[299] := -0.899879974464853;
cos_table[300] := -0.853204385517229;
cos_table[301] := -0.0220966192786839;
cos_table[302] := 0.829326676820903;
cos_table[303] := 0.918270850887274;
cos_table[304] := 0.162961039470883;
cos_table[305] := -0.7421744001017;
cos_table[306] := -0.964958118933387;
cos_table[307] := -0.300563793350083;
cos_table[308] := 0.640167497718337;
cos_table[309] := 0.992331743668192;
cos_table[310] := 0.432150760861815;
cos_table[311] := -0.525347638515573;
cos_table[312] := -0.999843841806507;
cos_table[313] := -0.555088227956658;
cos_table[314] := 0.400012942756024;
cos_table[315] := 0.987344058653017;
cos_table[316] := 0.666915600394842;
cos_table[317] := -0.266671985227481;
cos_table[318] := -0.955082577452527;
cos_table[319] := -0.765394652556692;
cos_table[320] := 0.127993586101478;
cos_table[321] := 0.903705111970614;
cos_table[322] := 0.848554325543618;
cos_table[323] := 0.0132466055205879;
cos_table[324] := -0.83423998252822;
cos_table[325] := -0.914730177935375;
cos_table[326] := -0.154221666243094;
cos_table[327] := 0.748077534163434;
cos_table[328] := 0.962597699596405;
cos_table[329] := 0.292109979267175;
cos_table[330] := -0.646942308866107;
```

```
    cos_table[331] := -0.991198821755207;
    cos_table[332] := -0.424151709070136;
    cos_table[333] := 0.532858528858193;
    cos_table[334] := 0.999961092757309;
    cos_table[335] := 0.547704039532204;
    cos_table[336] := -0.408109581772219;
    cos_table[337] := -0.988709135689029;
    cos_table[338] := -0.660294069919136;
    cos_table[339] := 0.275192318632293;
    cos_table[340] := 0.957668158547592;
    cos_table[341] := 0.759668310007225;
    cos_table[342] := -0.136767079363879;
    cos_table[343] := -0.907459446701534;
    cos_table[344] := -0.843837783705451;
    cos_table[345] := -0.00439555392789772;
    cos_table[346] := 0.83908792785983;
    cos_table[347] := 0.911117838425468;
    cos_table[348] := 0.145470210177922;
    cos_table[349] := -0.75392205843696;
    cos_table[350] := -0.960161863414609;
    cos_table[351] := -0.283633279182166;
    cos_table[352] := 0.653666433888477;
    cos_table[353] := 0.989988242179262;
    cos_table[354] := 0.416119426175127;
    cos_table[355] := -0.540327671221366;
    cos_table[356] := -0.999999999545659;
    cos_table[357] := -0.54027694002395;
    cos_table[358] := 0.416174246541013;
    cos_table[359] := 0.98999675012204;
    cos_table[360] := 0.653620807244793;

    return cos_table[degrees];
}

double func abs(double a) {
    if (a < 0) then {
        a := a * -1.0;
    }

    return a;
}

wait(1234);

setSpeed(1, sin(getTime()%360 + 1)*10, 1000);
setSpeed(2, cos(getTime()%360 + 1)*10, 1000);
```

```
wait(2200);

setGain(1, getSpeed(1));
setGain(2, getSpeed(2));

{
    while(abs(getSpeed(2) - getSpeed(1)) >= 9)
    {
        setSpeed(1, sin(getTime()%360 + 1)*10, 1000);
        wait(1);
    }
} || {
    while(abs(getSpeed(2) - getSpeed(1)) >= 9)
    {
        setSpeed(2, cos(getTime()%360 + 1)*10, 1000);
        wait(1);
    }
}

setGain(3, getSpeed(1));
setGain(4, getSpeed(2));
setGain(5, abs(getSpeed(2)-getSpeed(1)));

setSpeed(1, 1000, 1000);
setSpeed(2, 1000, 1000);
```

E.2.9 parallel3.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/parallel3.spinner,v 1.2 2003/12/17 14:07:48 me1
//
//    File name: parallel3.spinner
//   Created By: Marc Eaddy
//   Created on: 11/30/2003
//
//      Purpose: Spinner program that excercizes waitUntil
//
//  Description:
//
///////////////////////////////////////////////////////////////////////
/*
 * $Log: parallel3.spinner,v $
 * Revision 1.2  2003/12/17 14:07:48  me133
 * o Removed sin and cosine functions
 * o Modified algorithm slightly so that it would terminate
 *
 * Revision 1.1  2003/12/02 05:33:46  me133
 * o Created
 */

createDiscs(2);

setFile(all, "Aeolean-Harp");

double func abs(double a) {
    if (a < 0) then {
        a := a * -1.0;
    }

    return a;
}

double func getAngle(double discNum) {
    return getPosition(discNum) % 360;
}

seek(1, 10, 1);
wait(1);

{
    setSpeed(1, 10, 1000);
    waitUntil(abs(getAngle(2) - getAngle(1)) < 5, 5000);
    setSpeed(1, getSpeed(2), 2000);
    wait(2000);
} || {
    seek(2, 10, 500);
    wait(3000);
}

setSpeed(1, 0, 1000);
setSpeed(2, 0, 1000);
```

E.2.10 parallel4.spinner

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
```

```
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/parallel4.spinner,v 1.2 2003/12/12 23:54:28 gk2
//
//    File name: test2.spinner
//   Created By: Gaurav Khandpur
//   Created on: 12/12/2003
//
//      Purpose: Sample Spinner program for Final Demo
//
//  Description:
// Disc no. 1,3,5 go 3 revolutions clockwise in 3 seconds all together and
// then discs 2,4,6 go 3 revolutions anticlockwise all together. Then revesre
// the motion of 1,3,5 and 2,4,6
//  Status: Works but gives.
// 1, --------------- Time: 6000;
// 1, 2-line 2.1938006966593093E-13 3000;
// 1, 4-line 2.1938006966593093E-13 3000;
// 1, 6-line 2.1938006966593093E-13 3000;
//
////////////////////////////////////////////////////////////////////////
createDiscs(6);

/*This function does the simulation for rotating 3 discs -a,b,c 3 revolutions clockwise or anticlockwise
 *in 'time' seconds simulaneously.
 */
void func simul3discs(double a,double b,double c,boolean clockwise,double time) {
if(clockwise) then { //clockwise
seek(a,getPosition(a)+3,time);
seek(b,getPosition(b)+3,time);
seek(c,getPosition(c)+3,time);
} else { //anticlockwise
seek(a,getPosition(a)-3,time);
seek(b,getPosition(b)-3,time);
seek(c,getPosition(c)-3,time);
}
}

{
simul3discs(1,3,5,true,3000);
wait(3000);
}||
{
wait(3000);
simul3discs(2,4,6,false,3000);
wait(3000);
}
```

```
//CHECK HERE
{
simul3discs(2,4,6,true,3000);
wait(3000);
}||
{
wait(3000);
simul3discs(1,3,5,false,3000);
wait(3000);
}
```

E.2.11 parallelSimple.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/parallelSimple.spinner,v 1.2 2003/12/17 14:08:3
//
//    File name: parallelSimple.spinner
//   Created By: Marc Eaddy
//   Created on: 12/9/2003
//
//      Purpose:
//
//  Description:
//
///////////////////////////////////////////////////////////////////////
/**
 * $Log: parallelSimple.spinner,v $
 * Revision 1.2  2003/12/17 14:08:32  me133
 * o Now performs testing of nested parallel statements
 *
 * Revision 1.1  2003/12/11 15:03:52  me133
 * no message
 *
 */

// Initialization
createDiscs(6);

{
    wait(2);
    wait(3);
    waitUntil(getTime() > 5, -1);
    setGain(1, 200);
    for(double i := 0; i < 2; ++i)
    {
        commentVar(i);

        {
            commentNumTextNum(0, " branch started at: ", getTime());
            wait(2);
            setGain(2, 210);
            commentNumTextNum(0, " branch ended at: ", getTime());
        } || {
            commentNumTextNum(1, " branch started at: ", getTime());
            wait(3);
            setGain(3, 230);
            commentNumTextNum(1, " branch ended at: ", getTime());
        }
    }

    setGain(4, 250);

} || {
    wait(70);
    setGain(5, 300);
}

//setFile(all, "Aeolean-Harp");
```

E.2.12 testWaitUntil.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/testWaitUntil.spinner,v 1.6 2003/12/17 15:14:0
//
//    File name: testWaitUntil.spinner
//   Created By: William Beaver
//   Created on: 12/7/2003
//    $Revision: 1.6 $
//
//      Purpose: testing Spinner waitUntil
//
//  Description: since there is no way to output success/fail, we'll
// use setGain to set values: 1=pass 0=fail
// and output wait statement where duration=testnumber
// look through output for 1, 1-gain 0.0; for test failures
// immediately preceeding the gain is the failed test number
//                  NOTE: be sure to check the SPEED in sec i.e. 1 rev/sec NOT .001 rev/ms
///////////////////////////////////////////////////////////////////////
/*
 * $Log: testWaitUntil.spinner,v $
 * Revision 1.6  2003/12/17 15:14:02  wmb2013
 * simplified test
 *
 * Revision 1.5  2003/12/17 14:10:46  me133
 * o Trivial changes
 * o Fixed spelling mistake
 * o waitUntil now takes a timeout param
 *
 * Revision 1.4  2003/12/08 01:05:33  wmb2013
 * fixed filetype flag to ascii
 */

createDiscs(6);
double testTime;

//test number 10  = waiting for another disc to reach position
wait(10);
testTime:= getTime();
{
setSpeed(1,1,0);
}
||
{
waitUntil(getPosition(1)>=.5,-1);
if (getTime()-testTime =500) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
}

//test number 20  = waiting for signal
wait(20);
boolean signal;
signal:= false;
testTime:= getTime();
{
seek(1,2,1500);
//waitUntil(getPosition(1)>=2,-1);
wait(1500);
signal:=true;
}
||
{
waitUntil(signal,-1);
if (getTime()-testTime =1500) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
}
```

E.3 Simulator

E.3.1 testAllBuiltIn.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/testAllBuiltIn.spinner,v 1.4 2003/12/06 06:56:0
//
//    File name: testAllBuiltIn.spinner
//   Created By: William Beaver
//   Created on: 12/5/2003
//    $Revision: 1.4 $
//
//      Purpose: testing Spinner built in functions
//
//  Description: since there is no way to output success/fail, we'll
// use setGain to set values: 1=pass 0=fail
// and output wait statement where duration=testnumber
// look through output for 1, 1-gain 0.0; for test failures
// immediately preceeding the gain is the failed test number
//                  NOTE: be sure to check SPEED in ms i.e. 1 rev/sec = .001 rev/ms
///////////////////////////////////////////////////////////////////////
/*
 * $Log: testAllBuiltIn.spinner,v $
 * Revision 1.4  2003/12/06 06:56:03  wmb2013
 * minor typo
 *
 *
 */

//import("DiscLib1.spinner"); //test import
```

```
//setup();

//test number 0  = time Does not move
// if (getTime()=0) then { //pass
// //setGain(1,1);
// } else { //fail
// setGain(1,0); //look for this lone setGain if time 0 failed
// }

//test number 1000  = discs size on emptydiscs
if (getDiscsSize()=null) then { //pass
//setGain(1,1);
} else { //fail
setGain(1,0);// this will throw a compile error since disc 1 not defined
}

//test number 1001  = createDiscs and Get Discs size
createDiscs(6);
wait(1001);
if (getDiscsSize()=6) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 1002 = simulation time moved with prior and present waits
wait(1002);
if (getTime()= 2003) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//now we'll test the easy ones...

//test number 1  =getSpeed on stopped disc
wait(1);
if (getSpeed(1)=0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 2  = getPosition on stopped disc in 0 position
wait(2);
```

```
if (getPosition(1)=0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 3  = getGain on disc with unitialized gain
wait(3);
if (getGain(2)=null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 4  = setGain on disc with unitialized gain
wait(4);
setGain(2,0.5);
if (getGain(2)=0.5) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 5  = isFileset on disc without file set
wait(5);
if (not isFileSet(1)) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 6  = setFile and isFileset
wait(6);
setFile(1,"10-beeps");
if (isFileSet(1)) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 7  = getReverb send with reverb null
wait(7);
if (getReverb("send")= null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 8  = getReverb return with reverb null
wait(8);
if (getReverb("return")= null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 9  = set reverb send and getReverb send
wait(9);
setReverb("send",1.25);
if (getReverb("send")= 1.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 10  = setReverb return and getReverb return
wait(10);
setReverb("return",2.25);
if (getReverb("return")= 2.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 11  = getSpinMult on null
wait(11);
if (getSpinMult(1)=null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 12  = setSpinMult get Spin mult
wait(12);
setSpinMult(1,1.5);
if (getSpinMult(1)=1.5) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 13  = getSampleStart on null
```

```
wait(13);
if (getSampleStart(1)=null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 14  = setSampleStart getSampleStart
wait(14);
setSampleStart(1,3.5);
if (getSampleStart(1)=3.5) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 15  = getLength on null
wait(15);
if (getLength(1)=null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 16  = setLength getLength
wait(16);
setLength(1,10);
if (getLength(1)=10) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//setslide tests
//test number 17  = getSlide up with slide null
wait(17);
if (getSlide(1,"up")= null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 18  = getSlide down with slide null
wait(18);
if (getSlide(1,"up")= null) then { //pass
setGain(1,1);
```

```
} else { //fail
setGain(1,0);
}

//test number 19  = set slide up and getslide up
wait(19);
setSlide(1, "up", 1.25);
if (getSlide(1, "up")= 1.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 20  = set slide down and getslide down
wait(20);
setSlide(1, "down", 2.25);
if (getSlide(1, "down")= 2.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//setsub tests
//test number 21  = getSub send with sub null
wait(21);
if (getSub("send")= null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 22  = getSub return with sub null
wait(22);
if (getSub("return")= null) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 23  = setSub send and getSub send
wait(23);
setSub("send", 1.25);
if (getSub("send")= 1.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
```

```
}

//test number 24  = setSub return and getSub return
wait(24);
setSub("return", 2.25);
if (getSub("return")= 2.25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 25  = getSize for an array
// wait(25);
// double myArray[4];
// if (getSize(myArray)= 4) then { //pass
// setGain(1,1);
// } else { //fail
// setGain(1,0);
// }

// Now for the fun stuff...motion


//test number 26 = seek a disc to one revolution in one second
//you will see a few setgains on this test so look for them
wait(26);
double runTime;
runTime:= getTime(); //grab current time so you can check things
seek(1,1,1000);
wait(250);
//check to see if disc moved .25 revolutions
if (getPosition(1)= .25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
//be sure to check speed in ms i.e. 1 rev/sec = .001 rev/ms
if (getSpeed(1)= 1) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1));
}
wait(500);
//check to see if disc moved .75 revolutions
if (getPosition(1)= .75) then { //pass
```

```
setGain(1,1);
} else { //fail
setGain(1,0);
}
if (getSpeed(1)= 1) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
wait(250);
//check to see if disc moved full 1 revolution
if (getPosition(1)= 1) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
//disc should of stopped now
if (getSpeed(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1));
}

//test 27 = is the clock working?  i've moved forward 1000ms
if ((getTime()-runTime) = 1000) then {//pass
wait(27); //output test number
setGain(1,1);
} else { //fail
wait(27);//output test number
setGain(1,0);
}

//test number 28 = seek disc back to zero position in 1 second
//you will see a few setgains on this test so look for them
wait(28);
runTime:= getTime(); //grab current time so you can check things
seek(1,0,1000);
wait(250);
//check to see if disc moved -.25 revolutions
if (getPosition(1)= .75) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
if (getSpeed(1)= -1) then { //pass
```

```
setGain(1,1);
} else { //fail
setGain(1,0);
}
wait(500);
//check to see if disc moved .75 revolutions
if (getPosition(1)= .25) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
if (getSpeed(1)= -1) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
wait(250);
//check to see if disc moved full 1 revolution
if (getPosition(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
//disc should of stopped now
if (getSpeed(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//now for some setSpeed tests

//test number 29 = set speed to 2 rev/1000ms with ramp of 500ms
wait(29);

runTime:=getTime(); //grab the time

setSpeed(1,2,500); //set the speed

//this would be a great place to test what happens at boundary
//when command has been issued but time has not moved.  i bet the
//speed has changed
if (getPosition(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
```

```
}
//disc should not be moving
if (getSpeed(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
//advance time .25 secs.
wait(250);
if (getSpeed(1)= 1) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
//advance time .25 secs.
wait(250);
if (getSpeed(1)= 2) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1));
}
//advance time 1 secs. speed should be same
wait(1000);
if (getSpeed(1)= 2) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

//test number 30 = stop disc
wait(30);
setSpeed(1,0,1);
wait(1);
if (getSpeed(1)= 0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}

/* Write more tests like:
   issue consecutive seeks before advancing time to ensure
         last is the 'official' (i.e. no additive)
   issues consecutive setSpeeds before advancign time to ensure
         last is the official (i.e. no additive)
   test simple waitUntil;
   IMPORTANT: test stepping past your target condition with time.  does it recover?
 */
```

E.3.2 testSpeedSeek.spinner

```
//////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/testSpeedSeek.spinner,v 1.2 2003/12/17 14:10:12
//
//    File name: testSpeedSeek.spinner
//   Created By: William Beaver
//   Created on: 12/6/2003
//    $Revision: 1.2 $
//
//      Purpose: testing Spinner seek and speed commands
//
//  Description: since there is no way to output success/fail, we'll
// use setGain to set values: 1=pass 0=fail
// and output wait statement where duration=testnumber
// look through output for 1, 1-gain 0.0; for test failures
// immediately preceeding the gain is the failed test number
//                  NOTE: be sure to check SPEED in sec i.e. 1 rev/sec not .001 rev/ms
//////////////////////////////////////////////////////////////////////
/*
 * $Log: testSpeedSeek.spinner,v $
 * Revision 1.2  2003/12/17 14:10:12  me133
 * o Trivial changes
 *
 * Revision 1.1  2003/12/06 18:37:41  wmb2013
 * created
 */

/* Write more tests like:
   test simple waitUntil;
*/

if (getDiscsSize()=null) then {createDiscs(6);}

//test number 10  = seek with wait past condition.
wait(10);
seek(1,2,1000);
wait(1500);
if (getPosition(1)=2) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getPosition(1)); //output position if fail
}
if (getSpeed(1)=0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1)); //output Speed if fail
}

//test number 20  = setSpeed with no Ramp time.
wait(20);
setSpeed(1,2,0);
wait(1000);
if (getPosition(1)=4) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getPosition(1)); //output position if fail
}
if (getSpeed(1)=2) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1)); //output Speed if fail
}

//test 30  reset with seek after speed
wait(30);
seek(1,0,0); //get disc one to position 0 now.
if (getPosition(1)=0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getPosition(1)); //output position if fail
}
if (getSpeed(1)=0) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1)); //output Speed if fail
}

//test 40 = setSpeed with odd ramp and wait past ramp
wait(40);
```

```
setSpeed(1,2.5,2300);
wait(2500);//step past ramp
//not worried about position, just check speed
if (getSpeed(1)=2.5) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
setGain(1,getSpeed(1)); //output Speed if fail
}
```

E.3.3 testWaitUntil.spinner

```
///////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/testWaitUntil.spinner,v 1.6 2003/12/17 15:14:0
//
//    File name: testWaitUntil.spinner
//   Created By: William Beaver
//   Created on: 12/7/2003
//    $Revision: 1.6 $
//
//      Purpose: testing Spinner waitUntil
//
//  Description: since there is no way to output success/fail, we'll
// use setGain to set values: 1=pass 0=fail
// and output wait statement where duration=testnumber
// look through output for 1, 1-gain 0.0; for test failures
// immediately preceeding the gain is the failed test number
//                  NOTE: be sure to check the SPEED in sec i.e. 1 rev/sec NOT .001 rev/ms
///////////////////////////////////////////////////////////////////////
/*
 * $Log: testWaitUntil.spinner,v $
 * Revision 1.6  2003/12/17 15:14:02  wmb2013
 * simplified test
 *
 * Revision 1.5  2003/12/17 14:10:46  me133
 * o Trivial changes
 * o Fixed spelling mistake
 * o waitUntil now takes a timeout param
 *
 * Revision 1.4  2003/12/08 01:05:33  wmb2013
 * fixed filetype flag to ascii
 */
```

```
createDiscs(6);
double testTime;

//test number 10  = waiting for another disc to reach position
wait(10);
testTime:= getTime();
{
setSpeed(1,1,0);
}
||
{
waitUntil(getPosition(1)>=.5,-1);
if (getTime()-testTime =500) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
}

//test number 20  = waiting for signal
wait(20);
boolean signal;
signal:= false;
testTime:= getTime();
{
seek(1,2,1500);
//waitUntil(getPosition(1)>=2,-1);
wait(1500);
signal:=true;
}
||
{
waitUntil(signal,-1);
if (getTime()-testTime =1500) then { //pass
setGain(1,1);
} else { //fail
setGain(1,0);
}
}
```

E.3.4 test_all_constructs.spinner

```
/////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
```

```
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/test_all_constructs.spinner,v 1.15 2003/12/17 1
//
//    File name: test_all_constructs.spinner
//   Created By: Marc Eaddy
//   Created on: 11/23/2003
//
//      Purpose: Simple test of every Spinner programming construct
//
//  Description:
//
///////////////////////////////////////////////////////////////////////
/**
 * $Log: test_all_constructs.spinner,v $
 * Revision 1.15  2003/12/17 14:09:44  me133
 * o Added in the time comments
 * o Changed the prefix for all spin script cmds to "1"
 * o Added test for counting backwards in a for statement
 *
 * Revision 1.14  2003/12/02 05:33:10  me133
 * o Added trivial waitUntil test
 *
 * Revision 1.13  2003/12/02 02:17:56  me133
 * o Added variable and array test
 *
 * Revision 1.12  2003/12/01 22:51:50  me133
 * o Fixed comments
 *
 * Revision 1.5  2003/12/01 22:36:26  me133
 * o Per William's request, added readfile command
 * o Got rid of extra newlines
 *
 * Revision 1.4  2003/11/30 20:12:36  me133
 * o For statement now uses a variable
 * o Function example now uses parameters and returns a Double
 *
 * Revision 1.3  2003/11/24 05:45:02  me133
 * o Comments
 *
 * Revision 1.2  2003/11/24 05:19:15  me133
 * o Now tests function calling
 *
 * Revision 1.1  2003/11/24 03:40:10  me133
 * o Created
 */
                                // 1, -------------- Time: 0;
// Initialization
```

```
createDiscs(6);

setFile(all, "Aeolean-Harp");   // 1, all-readfile Aeolean-Harp;

// Builtin function call and Addition
setGain(1, 1+2);  // 1, 1-gain 3.0;

// Subtraction (positive result)
setGain(1, 6-2);  // 1, 1-gain 4.0;

// Subtraction (negative result)
setGain(1, 3-19);  // 1, 1-gain -16.0;

// Multiplication
setGain(1, 2*3);  // 1, 1-gain 6.0;

// Division
setGain(1, 10/5);  // 1, 1-gain 2.0;

// Modulo
setGain(1, 6%5);  // 1, 1-gain 1.0;

// Negative numbers
setGain(1, -2 * 3);  // 1, 1-gain -6.0;

// Associativity
setGain(1, 1 - 3 + 4);  // 1, 1-gain 2.0;

// Precedence
setGain(1, 1 - 3 * 2 + 4);  // 1, 1-gain -1.0;

// if-then and Equality
if (1 = 1) then {
setGain(2, 1); // 1, 2-gain 1.0;
}

// if-then and Equality and "and"
if (1 = 1 and 2 = 2) then {
setGain(2, 2); // 1, 2-gain 2.0;
}

// if-then and Equality and "or"
if (1 = 2 or 2 = 2) then {
setGain(2, 3); // 1, 2-gain 3.0;
}
```

```
// if-then-else and Equality
if (1 = 2) then {
setGain(2, 0.1);
}
else {
setGain(2, 4); // 1, 2-gain 4.0;
}

// if-then-else-if-then and Equality
if (1 = 2) then {
setGain(2, 0.2);
}
else if (1 = 1) then {
setGain(2, 5); // 1, 2-gain 5.0;
}

// if-then-else-if-then-else and Equality
if (1 = 2) then {
setGain(2, 0.3);
}
else if (1 = 3) then {
setGain(2, 0.4);
}
else {
setGain(2, 6); // 1, 2-gain 6.0;
}

// if-then and Inequality
if (1 != 2) then {
setGain(3, 1); // 1, 3-gain 1.0;
}
else {
setGain(3, 0.1);
}

// if-then and Equality and "and"
if (1 != 2 and 2 != 3) then {
setGain(3, 2); // 1, 3-gain 2.0;
}
else {
setGain(3, 0.2);
}

// if-then and Equality and "or"
if (1 != 1 or 2 != 3) then {
setGain(3, 3); // 1, 3-gain 3.0;
```

```
}
else {
setGain(3, 0.3);
}

// LT
if (1 < 2) then {
setGain(4, 1); // 1, 4-gain 1.0;
}
else {
setGain(4, 0.1);
}

// LE
if (1 <= 1) then {
setGain(4, 2); // 1, 4-gain 2.0;
}
else {
setGain(4, 0.2);
}

if (1 <= 2) then {
setGain(4, 3); // 1, 4-gain 3.0;
}
else {
setGain(4, 0.3);
}

// GT
if (2 > 1) then {
setGain(5, 1); // 1, 5-gain 1.0;
}
else {
setGain(5, 0.1);
}

// GE
if (1 >= 1) then {
setGain(5, 2); // 1, 5-gain 2.0;
}
else {
setGain(5, 0.2);
}

if (2 >= 1) then {
setGain(5, 3); // 1, 5-gain 3.0;
```

```
}
else {
setGain(5, 0.3);
}

// Not
if (not 1 = 2) then {
setGain(6, 1); // 1, 6-gain 1.0;
}
else {
setGain(6, 0);
}

// while // 1, 1-gain 0.0;
while (getTime() < 3) { // 1, --wait-- 1;
setGain(1, getTime()); // 1, --------------- Time: 1;
wait(1); // 1, 1-gain 1.0;
} // 1, --wait-- 1;
                                // 1, --------------- Time: 2;
                                // 1, 1-gain 2.0;
                                // 1, --wait-- 1;
                                // 1, --------------- Time: 3;

// for
for(double b := 2; getTime() < 6; wait(1)) {// 1, 2-gain 3.0;
setGain(b, getTime()); // 1, --wait-- 1;
}                               // 1, --------------- Time: 4;
// 1, 2-gain 4.0;
// 1, --wait-- 1;
                                // 1, --------------- Time: 5;
// 1, 2-gain 5.0;
// 1, --wait-- 1;
                                // 1, --------------- Time: 6;

// repeat-until // 1, 3-gain 6.0;
repeat { // 1, --wait-- 1;
setGain(3, getTime()); // 1, --------------- Time: 7;
wait(1); // 1, 3-gain 7.0;
} // 1, --wait-- 1;
until (getTime() >= 9); // 1, --------------- Time: 8;
                                // 1, 3-gain 8.0;
                                // 1, --wait-- 1;
                                // 1, --------------- Time: 9;

// Function definition
double func testFunc(double a, double b) {
```

```
setGain(a, getTime());
wait(b);
return 5;
        //return 0; // Should be ignored!
}

//testFunc(4, 1); // 1, 4-gain 9.0;
// 1, --wait-- 1;
testFunc(testFunc(4, 1), 1); // 1, -------------- Time: 10;
                                // 1, 5-gain 10.0;
// 1, --wait-- 1;
                                // 1, -------------- Time: 11;

// Variable decl with initial value
double nCount := 1.0;
setGain(1, nCount);             // 1, 1-gain 1.0;
nCount := 2;
setGain(1, nCount);             // 1, 1-gain 2.0;

// Variable decl without initial value
double nCount2;
setGain(2, nCount2);            // 1, 2-gain 0.0; (Should be compile error!)
nCount2 := 3;
setGain(2, nCount2);            // 1, 2-gain 3.0;

// Array decl
double ary[5];

// Initialize array using for-loop
for(double i := 1; i <= 5; i++)
{
    ary[i] := i;
}

for(double i := 1; i <= 5; i++) // 1, 1-gain 1.0;
{                               // 1, 2-gain 2.0;
    setGain(ary[i], ary[i]);    // 1, 3-gain 3.0;
}                               // 1, 4-gain 4.0;
                                // 1, 5-gain 5.0;
// Trivial test for waitUntil
waitUntil(true);

wait(1);                        // 1, --wait-- 1;
                                // 1, -------------- Time: 12;

// for
```

```
for(double c := 6; c >= 1; --c) {// 1, 6-gain 6.0;
setGain(c, c);          // 1, 5-gain 5.0;
} // 1, 4-gain 4.0;
                                // 1, 3-gain 3.0;
                                // 1, 2-gain 2.0;
                                // 1, 1-gain 1.0;
```

E.4 End to End Testing

E.4.1 testCue01.spinner

```
////////////////////////////////////////////////////////////////////////
//  Copyright (C) 2003 William Beaver, Marc Eaddy, Gaurav Khandpur,
//  Sangho Shin.  All Rights Reserved.
//
//      $Header: /u/4/m/me133/cvs/Spinner/compiler/tests/testCue01.spinner,v 1.5 2003/12/17 15:15:47 wmb
//
//    File name: testCue01.spinner
//   Created By: William Beaver
//   Created on: 12/7/2003
//    $Revision: 1.5 $
//
//      Purpose: testing Spinner mimic cue1.spinsciprt
//
//  Description:
//
////////////////////////////////////////////////////////////////////////
/*
 * $Log: testCue01.spinner,v $
 * Revision 1.5  2003/12/17 15:15:47  wmb2013
 * timings now based on var RATE so you can speed it up/down
 *
 * Revision 1.4  2003/12/07 22:54:28  wmb2013
 * created
 *
 *
 */

double RATE;
RATE:=250;
if (getDiscsSize()=null) then {createDiscs(6);}
setGain(all, 0);
wait(25);
setGain(master, 0.3);
setReverb("send", 0.11);
setReverb("return", 0.1);
setSub("send", 0.2);
setSlide(all,"up", 0);
setSlide(all,"down", 0);
seek(all,0,0);
wait(100);
setSpinMult(all, 1);
setSampleStart(all, 0);
setFile(all, "Aeolean-Harp");
wait(8000);
setSlide(all,"up", 10000);
setSlide(all,"down", 10000);
setGain(all, 0.7);

{ //branch for disc1
wait(68*RATE);
seek(1,-2, 32*RATE);
wait(32*RATE);
seek(1, -10, 34*RATE);
wait(34*RATE);
}
||
{ //branch for disc2
seek(2,4, 32*RATE);
wait(32*RATE);
seek(2,8, 32*RATE);
wait(32*RATE);
seek(2,10, 16*RATE);
wait(16*RATE);
seek(2,12, 16*RATE);
wait(16*RATE);
seek(2,14, 16*RATE);
wait(16*RATE);
}
||
{ //branch for disc3
wait(68*RATE);
seek(3, -2.5, 40*RATE);
wait(40*RATE);
seek(3, -10.5, 32*RATE);
wait(32*RATE);
}
||
{ //branch for disc4
wait(2*RATE);
seek(4, 4, 32*RATE);
wait(32*RATE);
seek(4, 8, 30*RATE);
wait(30*RATE);
seek(4, 10, 16*RATE);
wait(16*RATE);
seek(4, 12, 15*RATE);
wait(15*RATE);
seek(4, 14, 16*RATE);
wait(16*RATE);
}
||
{ //branch for disc5
 wait(68*RATE);
seek(5,-3, 48*RATE);
wait(48*RATE);
seek(5,-15, 32*RATE);
wait(32*RATE);
seek(5, -15.5, 8*RATE);
wait(8*RATE);
}
||
{ //branch for disc6
wait(2*RATE);
seek(6, 4, 32*RATE);
wait(32*RATE);
seek(6, 8, 30*RATE);
wait(30*RATE);
seek(6, 10, 16*RATE);
wait(16*RATE);
seek(6, 12, 15*RATE);
wait(15*RATE);
seek(6, 14, 16*RATE);
wait(16*RATE);
}
```