WCL: Website Creation Language

Sarah Friedman
Toby S. Lazar
Miguel A. Maldonado
Matthew Mintz-Habib, *Project Leader*
{sf2179, tsl2103, mam2136, mm2571} @ columbia.edu

December 21, 2004

Contents

1 Introduction

Websites today comprise the single largest source of publicly accessible information worldwide. Additionally, they provide the easiest and most cost-effective method of communicating information to a broad and geographically dispersed audience. As such, the ability to quickly and efficiently publish web pages is critically important for anyone wanting to display his or her information in the public view.

Currently, the underlying language of most web pages is HTML (Hyper Text Markup Language) or some extension of that (such as DHTML and XHTML). In addition, scripting abilities, such as JavaScript and VB Script, have been introduced that extend the functionality and usefulness of web pages. Even entire languages such as Java and php can be embedded into web pages to improve their range of functionality. Nevertheless, creating web pages for large projects still remains a somewhat tedious, if not outright difficult, job for the inexperienced web designer. To ease this job for the typical computer programmer, we have created a new language, the Website Creation Language (WCL).

WCL is intended for the novice or experienced computer programmer, whether or not he or she is experienced creating web pages with HTML. The main goals are: ease-of-use, efficiency and robustness.

1.1 Ease of Use

The WCL syntax and semantics are intended to be intuitive to the typical computer programmer or anyone else familiar with basic programming constructs. The notions of variables, types, keywords, objects, and functions are all integral parts of this language. Basic methods for creating and manipulating tables, links, and images are intuitive and clearly identifiable. Any individual with some experience coding in other languages should quickly become comfortable creating websites with WCL.

1.2 Efficiency

In order to help programmers create web pages quickly, WCL facilitates the combination and reuse of web components. This allows the programmer to create templates that can be used to create innovative web designs with a consistent look and feel. Having primitive date types that relate directly to HTML constructs ensures that the resulting HTML code is valid in accordance with the W3C specification. This eliminates the tiring and time-consuming debugging process of web applications. It also thereby ensures that the HTML output is compatible with all major browsers.

1.3 Robustness

Because of its simplicity, WCL can be easily used to create websites from the most simple to the most complex. The ability of WCL to include non-HTML code (such as JavaScript and DHTML) allows for creating complex, while keeping the basic structure of the WCL file simple and succinct.

2 Language Tutorial

2.1 Hello, world

The best way to learn a new language is to starting writing some examples. We start with the time-honored tradition of creating the "Hello, world" program. We show the code for producing a simple web page that, when loaded, contains only "Hello, world".

```
WebPage hello;
Paragraph par;
par.add("Hello, world");
hello.add(par);
hello.save("hello.html");
```

In the first line, a WebPage object is created and named `hello`. In the second line a Paragraph object, named `par`, is created. The third line adds the text "`Hello, world`" to the `par` object. That object is then inserted in to the `hello` WebPage object in line four. Finally, in line five, the HTML web page is created from the `hello` object. Quite apparent from this simple example is the ease with which a web page can be created. We'll soon explain how to translate this code into a real HTML web page.

It is important to note here the structure of the WebPage object and the way that that the `WebPage.save` method operates. A WebPage object contains a list of references to the various objects that are inserted into it, not their content. It is therefore very easy to change the contents of a WebPage object by directly the object that it refers to. The `save` method moves through the list of object references, sequentially saving each object to HTML code. Since a reference to the object and not a copy of the object is stored in the WebPage list, when the `save` method is invoked, the newest version of the object is saved. If an object was changed from the time it was inserted before the time it was saved, the newer changes will be reflected in the resulting web page. In our example, the third and fourth lines could have been interchanged with no effect on the resulting web page.

As you can see from this simple example, the WCL is quite intuitive. As you will see later, the language for creating more complicated web pages is also quite intuitive.

2.2 Compiling and Creating Web Pages

In order to compile the WCL file and create the HTML web page(s), the following steps must be taken. Before starting, a WCL file must be created. It must end with the extension '.wcl' and may be empty. The WCL compiler must then be run on the WCL file. The compiler takes the WCL code and creates a single java source file that, when compiled and executed, creates the web page(s). This java source file should be compiled and executed like any java source file. The following example demonstrates how to create the "hello.html" web page described above.

```
$ java Main hello.wcl
$ javac hello.java
$ java hello
```

After running these commands, the web page file, `hello.html`, should now exist in the local directory. Try opening it up in any web browser. You should see a white page with only the words `Hello, world` at the top. When you do, congratulations! You have just created your first web page using the WCL language.

2.3 A Second Example: Tables, Arrays and While Loops

We now demonstrate some more concepts of the WCL language with the following example. In this example we introduce comments, the Table object, arrays, and the while loop.

```
WebPage index;
Table myTable;

/* Creating variables */
int i = 0;
int j = 0;
int num = 0;
int size = 10;
int tmpArray[size];

while (i < 5)           // five rows
{
 j = 0;
 while (j < size)       // creating each row array
 {
  if (j % 2 == 1)
   num = j * i;
```

```
  else
   num = j * i * 2;
  tmpArray[j] = num;
  j = j + 1;
 }
 /* Adding array to the table */
 myTable.add(tmpArray);
 i = i + 1;
}
index.add(myTable);
index.save("index.html");
```

In this simple example, the resulting web page contains only a single table. Each row in the table is created by creating an array of numbers and then inserting each array as a row in the table. Finally the table is added to the `index` WebPage object and the web page is created.

This example introduces a few concepts whose details we be clearly defined later. For now we note the existence of two ways of enclosing comments. In one way a comment is encased by an opening '/*' and closed by '*/'. In the other way, anything following two front slashes is a comment. That comment extends until the end of the line. The Table object, `while` loop statement and integer arrays here introduced, will be explained in detail later.

In the following section we provide a comprehensive language reference manual for the Website Creation Language.

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 White Space

Aside from its function as token separator, white space has no affect on the output or execution of the WCL compiler. White space consists of a combination of any of the following characters:

```
"\n"    new line
"\r"    carriage return
"\t"    tab
" "     space
```

3.1.2 Comments

Comments are ignored completely by the WCL compiler. WCL comment blocks can be bookended by a beginning "/∗" and a final "∗/". Alternatively, anything following "//" on a line is ignored. Nested comments do not exist and can therefore cause unexpected results if not used properly. For example, the code:

```
/*
Paragraph par;
/* Paragraph par2; */
par.add("Hello, world");
*/
```

will cause a compile-time error. The nested-comment "*/" terminator also ends the outer comment block. The fourth line of the code will be compiled, generating an error since the `par` object has not been defined. The stray "*/" will cause an error in this example since it does not have a corresponding comment opener, "/*".

3.1.3 Identifiers

An identifier consists of a letter followed by a sequence one or more letters, digits and underscores. Upper and lower case letters are considered as different characters. Since WCL has only one namespace, identifiers may not be a keyword, a reserved word or the name of another object or function in the same scope.

3.1.4 Keywords

WCL reserves the following identifiers as keywords. They cannot be used as identifiers or as function names.

```
bool       declare    else       false      float      for
function   global     if         Image      int        Link
new        Paragraph  return     string     Table      true
WebPage    while
```

Additionally, the following are words reserved for internal operations. They may not be used as identifiers or as function names.

```
abstract   boolean    break      byte       case       catch
class      const      continue   default    do         double
extends    final      finally    goto       implements import
instanceof interface  long       native     new        package
private    protected  public     short      static     strictfp
```

```
super      switch     synchronized  this    throw      throws
transient  try        void       volatile
```

3.1.5 Punctuation

WCL strictly adheres to the following punctuation conventions:

```
( )     surround comparisons
{ }     surround statement blocks
[ ]     index of arrays
;       end of statement
.       object reference
,       list elements separator
```

3.2 WCL File Structure

A WCL file is composed of three sections: global variables, function declarations, and code. In the first section, all variables that will have a global scope must be defined. This is done with the keyword `global` preceding the variable type. For example, to make a global variable counter, one would write:

```
global int counter;
```

This statement creates a variable of type int that can be used anywhere within the program, always referring to the same object. Although its scope is global, functions can have variables that use the same name, in which case, any local reference to counter will be to the local version.

Function declarations must follow the global variables declarations and precede any other code. A function declaration takes on the following format:

```
declare function return-type function-name(argument-type argument-name);
```

In the argument list, there may be zero or more argument-type argument-name pairs. Functions may not be defined until the third and final section of the code. Although they may be defined anywhere within that code, defining them at the very end, makes the WCL code easier to read, and is suggested.

The final section contains the rest of the code, where the web page objects and content are created. This section must conclude with a blank line.

3.3 Operators

WCL uses various operators for mathematics and logical operations. The following operators are supported:

```
+       -       *       /
<       >       ==      !=
&&      ||      %       =
```

Their usage is similar to their usage in many other programming languages. There are four operators for basic arithmetic operations (`+, -, *, /`), six operators for boolean expression evaluations (`<, >, ==, !=, &&, ||`), one operator for assignments (`=`), and one operator that returns the remainder in integer division (`%`).

The arithmetic operators take operands of type int and float, though they must be consistent. It is legal to add two integers, and it is legal to add two float, but it is illegal to add an integer and a float. With these operators, one can add, subtract, multiply or divide integers or floats. The plus operator is overloaded and can therefore also be used to concatenate strings.

3.4 Primitive Data Types

WCL supports four basic data types: booleans, integers, floating point numbers and strings. No data type exists for a single character, but one can instead be implemented by a string containing a single character. In WCL, there is no implicit casting between different data types, except when concatenating a boolean, integer or floating point number with a string, in which case variables of those types are converted to a string. No other implicit conversions exist.

3.4.1 Booleans

Booleans can take on the values only of `true` or `false`. Comparisons always have the values, when evaluated, of either true or false. The default value for a declared but undefined boolean is false. The operators available for evaluating boolean expressions are presented in Table 1.

For example, one could assign the value true or false to boolean variable depending on the truth value of another expression. When assigning expressions to boolean variables, one can use the boolean operation operators listed above in any logically-correct combination. For example, one could evaluate the line $(t < 7||t > 100)\&\&(m < 9||m > 1000)$ and its truth value would depend on the values of t and m.

&&	boolean and - returns true if both (boolean) arguments are true and false otherwise
——	boolean or - returns true if either (boolean) arguments true and false otherwise
==	boolean equals - returns true if (numeric) arguments are equal and false otherwise
!=	boolean equals - returns true if (numeric) arguments are equal and false otherwise
>	greater than - return true if first argument is greater than second and false otherwise
<	less than - return true if first argument is less than second and false otherwise

Table 1: Boolean Operators

\n	newline
\r	carriage return
\t	tab
\b	backspace
\f	formfeed
\"	double quote
\\	backslash

Table 2: Escape Sequences

3.4.2 Integers

Integers consist of one or more digits and are optionally signed. Because of the way they are stored in memory, integers can range in value from -2147483648 to 2147483647. The values of integers assigned values outside of these bounds are undefined. The default value for an integer that is declared but not yet defined is 0.

3.4.3 Floating Point Numbers

Floating point numbers consist of three parts: an integral part, a fractional part and an exponent. The integral is part consists of an optionally signed (+ or -) integer. The fractional part consists of a decimal followed by an integer. The exponent consists of an 'e' or 'E', an optional sign, and an integer. Either the fractional part or the exponent part may be omitted, but not both. Because of the way they are stored in memory, floating point numbers can range in value from 1.4E-45 to 3.4028235E+38 with positive or negative signings. Like integers, the default value of an undefined float is 0.

3.4.4 Strings

A string is defined as a sequence of zero or more characters surrounded by double quotes. In order to represent certain characters within a string, escape characters are available. Table 2 provides the list of escape sequences and the characters they represent.

Numerous strings can be concatenated together with the "+" operator. For example, the sequence of statements:

```
string h = "hello, ";
string w = "world";
string hw = h + w;
hw = h + "world";
```

creates the "hello, world" string and assign it to the variable `hw`. The repetition of lines 3 and 4, demonstrate how both string literals and string variables can be concatenated together. Additionally, numbers can also be concatenated with strings, in which case the number (integer or float) is converted to a string before being concatenated. Before being initialized, a string variable points to an empty string.

3.5 Arrays

As in many programming languages, WCL provides an option to group a collection of similar items into a single data structure called an array. An array declaration must include the type of item as well as the number of elements it contains. Later references to individual elements in the array are made with the `[]` operator. Since the indices begin from 0, the index of the last element in the array is one less than the number of elements in the array. If reference is made to an element outside of the array bounds, a runtime error will occur when executing the resultant WCL java file. The values of the array items start with the default value for the item type (0, false or "").

Below is sample code that creates a table with five rows, where each row contains a number from 1 to 5, along with the square of each number.

```
Table squares;
int row[2];
int i=1;
while(i<=5)
{
  row[0] = i;
  row[1] = i*i;
  squares.add(row);
  i = i + 1;
}
```

The later section that discusses tables will explain the add method used here.

3.6 Objects

Five WCL objects comprise the bulk of WCL source files: WebPage, Paragraph, Link, Image, and Table. These objects encapsulate the contents of the web page and with them a web page is generated. The methods of each object are explained here.

3.6.1 WebPage

The WebPage object is the object used to encapsulate the contents of an entire web page. Objects of types Paragraph, Link, Image and Table can be added to its contents and subsequently can be used to create HTML code for a web page. An instance of this object is declared:

```
WebPage web-page-name;
```

This statement creates an instance of the WebPage object that starts off empty. Objects (of types Paragraph, Link, Image and Table) can be added to its list of content, by using the add method. The syntax for the add method is:

```
WebPage.add(object-name);
```

For example, the following code would create a WebPage object named testpage and add to it a Paragraph object named testpar (for which we haven't yet provided code).

```
WebPage testpage;
WebPage.add(testpar);
```

Although the WebPage object has been created and edited, it is not output to HTML form until the save method is invoked. To create an HTML source file from the testpage object, the save method should be invoked as follows:

```
WebPage.save(html-source-name);
```

In our example, to output the testpage object to an HTML file named test.html, the code would be:

```
testpage.save("test.html");
```

It is important to note here the way in which the WebPage objects stores its content objects. The list that it contains is one of pointers to object, not copies of them. If an object is changed directly before the containing WebPage object's save method is invoked, the updated object will be saved. This makes changing individual items in a web page convenient. One could

save different HTML files by simply changing the contents of one item before invoking the save command. Continuing our example, one could save two web pages, the first containing one version of the testpar object and the other completely identical except for a small change to the testpar object. The code would look like:

```
WebPage testpage;
WebPage.add(testpar);
testpage.save("test1.html");
testpar.add("\nhello again");
testpage.save("test2.html");
```

In this example, test1.html and test2.html differ only in that test2.html has an extra line containing the words "hello again". In this example we have used two methods of the Paragraph object. We now provide some more details about these methods.

3.6.2 Paragraph

The Paragraph object is used as a container for plain text that is inserted into a WebPage object. Its declaration is quite simple:

`Paragraph` *`paragraph-name`*`;`

This statement creates an empty paragraph, ready for filling with text. The Paragraph object has only one method, the add method. This method appends its argument string to the existing paragraph object. Its syntax is simply:

`Paragraph.add(`*`text`*`);`

A copy of the string text is added to the object and therefore changing the value of the string object will have no effect on the output of the WebPage save method. Deleting contents from the paragraph object is not allowed. An example of the add method was demonstrated above.

`Paragraph.setProperty(`*`property, value`*`);`

The setProperty method can be used to set any property that is available in the HTML language. Each argument must be a string; numeric values should be enclosed by quotes. Example properties of HTML paragraphs include alignment, font size, color and formatting.

3.6.3 Link

At the heart of the HTML language sits the concept of links. A link provides a reference to another file on the Internet. That other file could be an image, a music clip, or even just

another web page. The Link object in WCL is the container for that link. It is initialized, like its fellow objects with the following statement:

```
Link link-name;
```

Both arguments must be of type string. The values of the text and the url cannot be specified in the Link declaration, but must be set using (and can later be changed with) the following set methods:

```
Link.setText(text);
```

The *text* declared here will be the text displayed on the web page that references the Internet file specified by:

```
Link.setProperty(property, value);
```

Link properties can easily be set using the setProperty method. Like the Paragraph setProperty method, both arguments must be of type string. This powerful method has the ability to change any property of a link that is allowed by HTML. Common examples of link properties include mouse over, font color, clicked-on color, etc.

```
Image.setURL(url);
```

The arguments of both of these methods must be strings.

3.6.4 Image

The Image object is used as a container for an image that can be displayed on the web page. Its declaration is similar to the declarations of the other objects:

```
Image image-name;
```

The Image object is initialized without pointing to a specific image. The image source which specifies the location on the local hard disk of the image can be set (or changed) with the following method:

```
Image.setImage(image-source );
```

If the image source is not set before the web page is saved into HTML code, the HTML code will contain no trace of the image, its name or properties. An important method that is used in addition to setImage is the setProperty method.

```
Image.setProperty(property, value);
```

This method is used to set any property of the image. Examples of some properties include alt, border, and alignment. Both the property and the value must be expressed as strings. Any of these properties may be omitted and the generated HTML code will not include specific values for these properties.

3.6.5 Table

A convenient way to display a collection of structured information is by using a table. Using a table, one can display numerous records by putting each record in a separate row. Columns can represent separate fields of individual records. In WCL, a Table object is declared as follows:

`Table` *`table-name`*`;`

In this declaration, an table object is created. It begins empty without any restriction on the numbers of columns it contains. To add information to the table, an array of information must be first created and then added. This array may be of types `int`, `float` or `string`. The following command is used to add an array to a table:

`Table.add(`*`array`*`);`

The array is added to the table as a single row. Each element in the array is placed in another column of that row. For instance, if the array contained four elements, the new row would have four columns. An example of creating a table and inserting rows was shown before in our discussion before about arrays.

`Table.setProperty(`*`property, value`*`);`

Table properties can also be set using the setProperty method. For the table example above, we could add borders to the table using the following command:

```
myTable.setProperty("border", "2");
```

This code creates a border around the table as well as a grid for the table elements.

3.7 Declarations

Both variables and functions have rules for their declarations. The following two sections define these rules.

3.7.1 Variable Declarations

All variables must be declared before being used. They assume the following format:

declaration-type identifier `(= value);`

The declaration type and identifier are both required, whereas the assignment of a value to the identifier is optional.

Aside from its type, an identifier also has a lexical scope in which it can be referenced. Since its scope depends largely on where it is declared, we discuss scoping rules here. A variable's scope begins from the point it is declared and terminates at the end of the section it is nested in. For example, the lexical scope of a variable declared within a function will begin with the its declaration and will terminate at the end of that function call. At the end of an identifier's scope, the content of the variable becomes discarded and the identifier may be reused.

Additionally, a variable's scope extends into nested blocks within its scope. For example, a while loop within a function will have access to any variable declared before the loop in the function. Global variables (which will soon be explained) are always in scope throughout the program and within any function. In some cases where there is more than one variable with the same name (this can happen when an inner scope has a variable with the same name as a variable in its outer scope), any reference to that variable name will be to the inner-most variable.

3.7.2 Function Declarations

Like variables, functions must too be declared before their use. As mentioned above, all functions must be declared in the section following the global variables and preceding any other code. By dividing the WCL file in to three divisions, functions can be declared and then used, even before they are defined. This allows for easily readable code, where a the details of the function can be hidden until the end of the file. Additionally, in this fashion, functions can call one another with ease.

Functions in WCL must have return types (void is not an available type) as well as argument types. Both in the declaration of a function and in its later definition, these must be types must be consistent. More will be said about function definitions in the following section.

3.8 Functions

Functions play an important, if somewhat behind-the-scenes role in WCL. Although WCL makes creating web page files easy, attending to the minute details of number values can be

sometimes tedious. In WCL, functions allow creating more structured, more understandable code.

For example, someone wanting to create a multiplication chart, may find it troublesome to create a new record for each number and type in (for example, for the number 2) 2, 4, 6, 8, etc. In the following example, we provide a function that takes a Table object and adds 10 rows to it, each with the values found in a standard multipilcation chart.

```
declare function int MultChart(Table t);

function int MultChart(Table t)
{
 int i=1;
 while(i<11)
 {
  int line[10];
  j=1;
  while(j<11)
  {
   line[j-1] = i*j;
   j = j + 1;
  }
  t.add(line);
  i = i + 1;
 }
 return 0;
}
```

In this example, the MultChart function is first declared (in the function declarations section of the WCL file) and later define (anywhere in the rest of the code). The function takes a Table object, makes ten rows and inserts them into the table. Nothing needs to be returned since the Table object is passed by reference. The actual object taken is modified and not just a copy of it. This call by value aspect is true only for the objects. Like in Java, the basic data types are all called by value.

As can be seen above, a function must be first declared and then defined. While the declaration must come at the beginning of the file, after the declaration of the global variables, its definition may come anywhere in the later code. Functions may be call other functions and may also call themselves recursively, as they have already been declared before they are defined.

Since functions must have a return type, the function is declared to return type int and the line `return 0;` returns a 0.

The format of a function declaration is:

`declare function` *function-type function-name* `(` *(argument-type argument-name)** `);`

If a function is declared and never defined, the compiler will produce and error and the code will not be compiled. Function definitions reiterate the function signature and define the statements the function will perform. A function definition looks like:

`function` *function-type function-name* `(` *(argument-type argument-name)** `)`
{
    statement
}

Functions are called by stating their name and then their arguments surrounded by parentheses.

3.9 Statements

A WCL file is composed of zero or more statements. Each statement must be terminated by a semicolon and more than one statement may be placed on a single line, so long as there is a semicolon separating each statement. Types of statements include declarations, assignments, function definitions, function calls, while loops, conditional statements, or statements blocks. Those have all been explained previously in this document. Statement blocks are statements included between { and }. They are usually used following and if or while statement. Although mathematical expressions have a value, other expressions do not. For example, the expression `a = 6 + 7;` does not have a value. This being the case, the assignment `b = a = 6 + 7;` is illegal.

3.9.1 Conditional Statements

In some cases, it may be desirable to execute a statement conditionally, or execute one of two statements depending on some condition. For this, we have included the if-else pair of conditional statements. An if statement is used as follows:

`if(`*condition*`)`
    statement
`else`
    statement

In this statement, the boolean condition is first evaluated. If it is true, the first statement is executed. If it is false, the statement following the else if executed. The else section is optional and may be omitted. Although braces are not required for the statement, they are necessary when more than one statement will be executed if the condition is true. For example, we could assign the value true or false to a variable, depending on the value of a different number:

```
if(index > 7)
   flag = true;
else
   flag = true;
```

Or, we could include a few statements in the block that follows the condition evaluation. In this case braces must enclose the entire statement block.

```
if(index > 7)
{
   flag = true;
   mypage.save("index1.html");
}
else
{
   flag = true;
   mypage.save("index2.html");
}
```

3.9.2 While Loops

Sometimes one statement needs to be repeated numerous time. Instead of writing the same line out for each iteration, a while loop can be used. A while loop checks a condition and performs the contents of its statement block each time that the condition is evaluated as true. Its format is:

```
while(condition)
   statement
```

Just like the if-else statements above, the while statement may be a single statement or a statement block containing numerous statements. If it is a statement block, the block must be enclosed by braces, as was seen in the examples before. Like in almost every programming language, sloppy programming here can lead to infinite loops. Infinite loops are best avoided.

Although WCL does not allow for loops, a while loop can be used to have the same functionality if done properly. A for loop that assigns every element in an array the value of

its index can be written like this:

```
int i=0;
while(i<10)
{
  array[i] = i;
  i = i+1;
}
```

The loop will iterate 10 times before i assumes the value of 10 and the while condition is evaluated as false. In each iteration, the array with index i is assigned the value of i and then i is incremented by 1.

4 Project Plan

4.1 Organization of Project and Division of Labor

We divided the parts of the project and assigned to each team member an assigned part. Miguel handled the syntax and grammar of the language. Matthew generated the AST walker and related code. Sarah was in charge of the backend, including the java functions that implemented the semantics of the language. Toby was assigned the documentation as well as general code testing.

4.2 Software Development Environment

We developed and tested WCL on a variety of platforms. Matthew and Sarah worked on Windows machines, Miguel worked with Mac OS X, while Toby used the Linux cluster available at Columbia.

4.3 Project Log

Table 3 provides a log of our group's progress throughout the semester:

5 Architectural Design

The general design of WCL is similar to most languages. The file is run through the lexer and parser. Its AST table is run through a semantic check. If all goes well, the AST is walked, and in doing so, HTML code is produced. Figure 1 displays this process.

September 9	Meeting to decide on language
September 14	Meeting to discuss language
September 28	White paper complete
October 21	Tentative LRM proposed
October 27	ANTLR grammar file created
October 28	ANTLR semantics file created
December 3	ANTLR code generation file created
December 9	Code infrastructure complete. Testing begins of language semantics and syntax. Test suites used in full force. Testing by other group members also begins.
December 17	Project nearing completion. Minor bugs are being tested for and fixed.
December 21	Project complete. Presentation and submission of project code and report.

Table 3: Project Timeline

The code output from the WCL compiler is java code. That code is then compiled and run, which in turn produces the final product, zero or more HTML files. Figure 5 displays that process.

Because WCL uses an intermediate java stage, come WCL functions use java's abilities. For example, when a user creates a loop in WCL, that same loop is converted into java before being compiled by the java compiler. Similarly, WCL does not directly evaluate comparisons, but leaves that to java.

6 Test Plan

Although no amount of testing can catch every small bug, our tests suites focused on testing numerous cases of each function of the language. With a regression test suite, we were able to systematically test the various additions to the language. With each modification to the language, the test suite provided us feedback about how the alteration affected other parts of the programming language code. In total, the regression test suite included more than 150 tests, with more than one hundred tests for semantic checking alone!

In addition to the test suites, independant tests were performed in order to catch overlooked mistakes. These extra tests were done by a fellow team member not involved in that particular part of the project. This enabled a fair, unbiased look at the code limitations. When bugs were found, a message was sent to the entire group detailing the exact code and error output. This enables a timely pinpointing and fixing of small bugs.

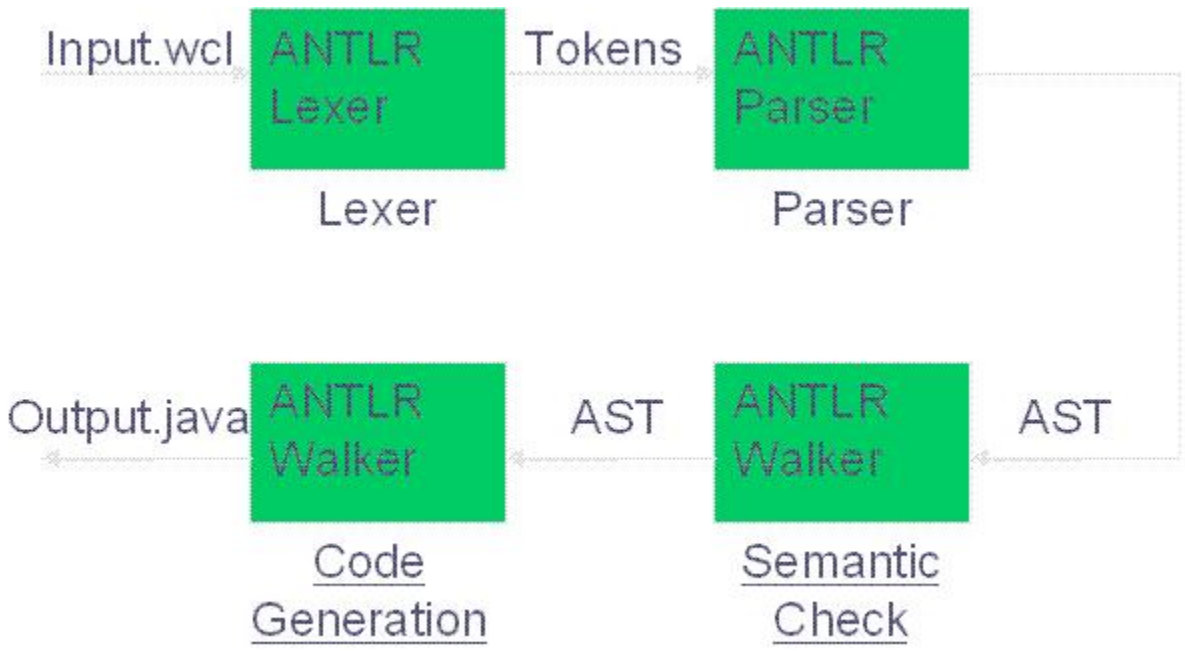

Figure 1: Architectural Design

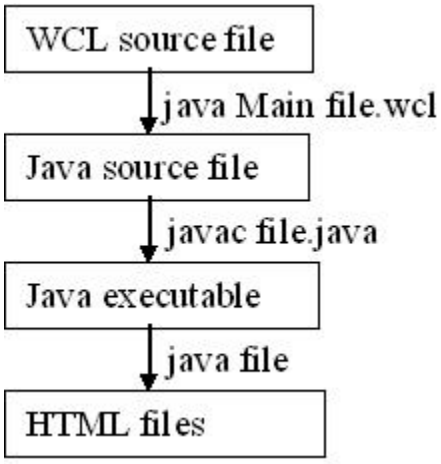

Figure 2: Compilition Process

6.1 Example 1: Hello World

Below is the source code and resulting HTML code for a page that contains just the string Hello, world:

```
WebPage w;
Paragraph p;
p.add("Hello, world");
w.add(p);
w.save("index.html");
```

```
<HTML>
<HEAD></HEAD>
<BODY>
<FONT > <P>Hello, world</P></FONT>
```

Also important to see is the intermediate java code used to produce the HTML file:

```
public class hello{
public static void main(String args[]) {
WebPage w= new WebPage();
Paragraph p= new Paragraph();
p.add("Hello, world");
w.add(p);
w.save("index.html");
}
}
```

6.2 Example 2: Tables

Below is pasted the WCL source file for an example including a table of integers that adds rows using arrays. Also provided is the intermediate java code and the final HTML file.

```
WebPage index;
Table myTable;

int tmpArray[5];
int num;
int size = 5;
int i=0;
while (i < 5) {  // five rows
```

```
   int j = 0;
   while (j < size) { // creating each row
   if (j % 2 == 1)
      num = j * i;
   else
      num = j * i * 2;
   tmpArray[j] = num;
   j = j + 1;
   }
   /* Adding array to the table */
   myTable.add(tmpArray);
   i = i + 1;
}
index.add(myTable);
index.save("index.html");


public class table{
public static void main(String args[]) {
WebPage index= new WebPage();
Table myTable= new Table();
int[] tmpArray= new int[5];
int num=0;
int size=5;
int i=0;
while (i<5){
int j=0;
while (j<size){
if (j%2==1){
num=j*i;
}
else {
num=j*i*2;
}
tmpArray[j]=num;
j=j+1;
}myTable.add(tmpArray);
i=i+1;
}index.add(myTable);
index.save("index.html");
```

```
}
}

<HTML>
<HEAD></HEAD>
<BODY>
<table ><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td>
<td>1</td><td>4</td><td>3</td><td>8</td></tr><tr><td>0</td><td>2</td><td>8</td>
<td>6</td><td>16</td></tr><tr><td>0</td><td>3</td><td>12</td><td>9</td><td>24</td>
</tr><tr><td>0</td><td>4</td><td>16</td><td>12</td><td>32</td></tr></table>
```

These examples were two our of the many tests that were tun to ensure that the language compiled and prooduced HTML files as expected.

6.3 Example 3: Failure

In cases of semantic failure, the compiler will display the line number of the error, the line itself and some detail about the error. Below is one example of error-catching in the compiler:

```
int t[10];
Table w;
t[w] = 9;

WclSemanticCheck: Error on line 3.
WclSemanticCheck: t[w] = 9;
WclSemanticCheck: Array indices can only be type ‘int’, but you’re
                  referencing type ‘Table’.
```

The above is the WCL source file along with the resulting error message.

6.4 Perl Code for Regression Test Suite

Below is pasted the perl script (written by Matthew) that ran the regression test suite:

```
#!/usr/bin/perl

#
# The regressive test suite controller.
#
# @author Miguel and Matt
#
```

```
use warnings;
use Cwd;

$cwd_dir = cwd();
$grammer_dir = "test/grammer";
$semantics_dir = "test/semantics";
$codegen_dir = "test/codegen";

# We'll use these later to print the results
$do_gram = 0;
$do_sem  = 0;
$do_code = 0;

# Check if there's a command line argument
if ($ARGV[0]) {
        # Only run subset of tests per command line argument
        if ($ARGV[0] =~ /^gram$/i) {
                # Only run grammer tests
                @test_dirs = ($grammer_dir);
                $do_gram = 1;
        } elsif ($ARGV[0] =~ /^sem$/i) {
                # Only run semantic tests
                @test_dirs = ($semantics_dir);
                $do_sem = 1;
        } elsif ($ARGV[0] =~ /^code$/i) {
                # Only run code generation tests
                @test_dirs = ($codegen_dir);
                $do_code = 1;
        } else {
                # Run all the tests
                @test_dirs = ($grammer_dir, $semantics_dir, $codegen_dir);
                $do_gram = 1;
                $do_sem  = 1;
                $do_code = 1;
        }
} else {
        # Run all the tests
        @test_dirs = ($grammer_dir, $semantics_dir, $codegen_dir);
        $do_gram = 1;
```

```
        $do_sem  = 1;
        $do_code = 1;
}

print "Start\n";

print "Create error.log...\n";
open(ERROR, ">error.log");

foreach $dir (@test_dirs) {
        $test_dir = $cwd_dir."/".$dir;
        opendir(DIR,"$test_dir") or die "opendir $test_dir failed: $!";

        if ($dir eq $grammer_dir) {
                print "\n***Running Grammer Tests***\n";
        } elsif ($dir eq $semantics_dir) {
                print "\n***Running Semantic Tests***\n";
        } elsif ($dir eq $codegen_dir) {
                print "\n***Running Code Generation Tests***\n";
        }

        print "Open dir: $test_dir\n";

        $success{$dir} = 0;
        $error{$dir} = 0;
        $total{$dir} = 0;
        $result = 0;

        while( defined ($file = readdir DIR) ) {

                if ($file !~ /\w.wcl|\w.result/) {
                        next;
                }
                if ($file =~ /result/) {
                        print "Store result: $file\n";
                        #$resultFile = $file;
                        $result = "";
                        open(RESULT, "$dir/$file") or warn "Could not
                         open $file: $!\n";
                        while (<RESULT>) {
```

```
                        $result = $result.$_;
                }
                close(RESULT);
                #print "$result ==>STORED\n";
}
if ($file =~ /wcl/) {
                print "Run test: $file\n";
                $total{$dir} = $total{$dir} + 1;
                $test = "";

                if ($dir eq $grammer_dir) {
                        open(TEST, "/opt/jdk-1.4.1/bin/java Main
                         $dir/$file gram|");
                        print(     "/opt/jdk-1.4.1/bin/java Main
                         $dir/$file gram\n");
                } elsif ($dir eq $semantics_dir) {
                        open(TEST, "/opt/jdk-1.4.1/bin/java Main
                         `$dir/$file sem|");
                        print(     "/opt/jdk-1.4.1/bin/java Main
                         $dir/$file sem\n");
                } else {
                        open(TEST, "/opt/jdk-1.4.1/bin/java Main
                         $dir/$file code|");
                        print(     "/opt/jdk-1.4.1/bin/java Main
                         $dir/$file code\n");
                }

                while(<TEST>) {
                        $test = $test.$_;
                }
                close(TEST);

                #print "$test ==>CHECK\n";
                if ($test eq $result) {
                        print "Passed test.\n";
                        $success{$dir} = $success{$dir} + 1;
                        $result="";
                } else {

                $error{$dir} = $error{$dir} + 1;
```

```
                        print ERROR "Error processeing: $file\n";
                        print ERROR "\tOld Result:\n\t\t$result\n";
                        print ERROR "\tNew Result:\n\t\t$test\n\n";
                       #Try to better identify the "diff"
                                    #$diff = "";
                                    #open(DIFF, "diff test/$resultFile
                                     test/$file.tmp|");
                                    #while (<DIFF>) {
                                    #    $diff = $diff.$_;
                                    #}
                                    #print "Diff: $diff\n";
                        $result="";
                        }
                }
        }
        closedir(DIR);

}

# Print results only for tests we actuallly ran...
if ($do_gram) {
        print ERROR  "\n*******Grammer Tests*******\n";
        print STDOUT "\n*******Grammer Tests*******\n";
        $str = "Tests Passed: \t".$success{$grammer_dir}."/".$total{$grammer_dir}."\n";
        print ERROR  $str;
        print STDOUT $str;
        $str = "Tests Failed: \t".$error{$grammer_dir}."/".$total{$grammer_dir}."\n";
        print ERROR  $str;
        print STDOUT $str;
}
if ($do_sem) {
        print ERROR  "\n*******Semantic Tests******\n";
        print STDOUT "\n*******Semantic Tests******\n";
        $str = "Tests Passed: \t".$success{$semantics_dir}."/".$total{$semantics_dir}."\
        print ERROR  $str;
        print STDOUT $str;
        $str = "Tests Failed: \t".$error{$semantics_dir}."/".$total{$semantics_dir}."\n"
        print ERROR  $str;
        print STDOUT $str;
}
```

```
if ($do_code) {
        print ERROR  "\n***Code Generation Tests***\n";
        print STDOUT "\n***Code Generation Tests***\n";
        $str = "Tests Passed: \t".$success{$codegen_dir}."/".$total{$codegen_dir}."\n";
        print ERROR  $str;
        print STDOUT $str;
        $str = "Tests Failed: \t".$error{$codegen_dir}."/".$total{$codegen_dir}."\n";
        print ERROR  $str;
        print STDOUT $str;
}

close(ERROR);

print "\nDone\n";
```

7 Lessons Learned

7.1 Matthew

Lessons Learned and advice for students:

- The project is going to take more time than you anticipate. You'll hear it over and over, but really, start early!
- The semantics turned out to require many more error handling cases than expected. Be sure to keep your language to a reasonable size to avoid the tedious task of handling too many error cases.
- For the semantic check, don't encode types as strings directly. Add a layer of abstraction through a new class.
- It also turned out that after all the semantic checking, it would have been very easy to interpret the AST, rather than generating java code as we did.
- The regressive test suite is your friend. Make sure you stay good buddies!

7.2 Miguel

- The Esterel compiler is a great resource. You can use it as a template to solve grammar problems you may face.

- People like to see flashy images during presentations. Be sure to include some interesting images, like Calvin and Hobbes comics.
- Macintosh rules! Make sure you test your compiler on a Mac.
- Working closely with team members, being in constant touch via email and IM is critically important, especially in the final week and days.

7.3 Sarah

One reason that working on this project was a very good experience for me is because it taught me how to work with a team of three other people. We all had different schedules and finding times that were convenient for all of us and deadlines that were possible for all of us to meet were sometimes challenging. I also learned a lot about compilers. I never really thought much about how they worked before this class, and now I see how much really goes into it. Every time we made a change to one thing, we had to test everything done previously because often a fix for one thing, caused a bug in some other part of the code. Using CVS was very helpful because, if necessary, we could go back to older versions or our code. Also, having a regression test script was helpful, so we could make sure that we weren't breaking things that worked previously when we made a change.

7.4 Toby

Working on the WCL project enabled me to see the complexity and detail of programming language design. From the language semantics to code generation, careful attention must be paid to ensure that each component works well together with the other components. The importance of working separately, yet being able to coordinate with other team members was made abundantly clear during the final testing stages. At that point, a single bug found in the production code needed to be checked by each member, and the team member responsible for it needed to be identified. Usually, this was quickly done with emails to the group. CVS remained integrally important to this coordination. Occasional lapses of checking in or updating finished code was often the cause of minor bugs. Most of all, I learnt both from what we did and from hindsight, the importance of designing at the outset the goals, and general structure of the language. Future language designers should spend much time in figuring out not only what, but how their language will work and be constructed.

A WCL ANTLR Files

A.1 WCL Grammar

```
class WclLexer extends Lexer;

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


PERIOD : '.' ;
COMMA : ',' ;
COLON : ':' ;
SEMICOLON  : ';' ;
POUND : '#' ;
PLUS : '+' ;
MINUS : '-' ;
SLASH : '/' ;
MODULO : '%';
TIMES : '*' ;
UNARY : '&' ;

LESS : '<';
LESSEQUAL : "<=";
MORE : '>';
MOREEQUAL : ">=";
EQUAL : '=';
EQUALTO : "==";
NOT : '!';
OR : "||" ;
AND : "&&";
NOTEQUAL : "!=";
LEFTPAREN : '(';
RIGHTPAREN :')';
```

```
LEFTBRACKET : '[';
RIGHTBRACKET : ']';
LEFTBRACE: '{';
RIGHTBRACE: '}';

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

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

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


/// Numeric constants
/**
  The two types of number are split to avoid a side-effect of LL(k) parsing:
  If the two rules were combined, the lookahead set for Number would include
  a period (e.g., from ".1") followed by end-of-token e.g., from "1" by
  itself), which collides with the lookahead set for the single-period rule.
 */

Number
  : ('0'..'9')+
      ( '.' ('0'..'9')* (Exponent)?
        { $setType(NUM_FLOAT); }
      | /* empty */ { $setType(NUM_INT); }
      )
  ;

// a couple protected methods to assist in matching floating point numbers
protected
Exponent
        :       ('e'|'E') ('+'|'-')? ('0'..'9')+
        ;

protected
ESC
   :  '\\'
      (  'n'
      |  'r'
```

```
      |  't'
      |  'b'
      |  'f'
      |  '"'
      |  '\''
      |  '\\'
        )
    ;

String
   :  '"' (ESC|~('"'|'\\'|'\n'|'\r'))* '"'
    ;

Comment
    : '/'
        ( ('*') => '*'
            ( // Prevent .* from eating the whole file
                options {greedy=false;}:
                ( ('\r' '\n') => '\r' '\n' { newline(); }
                | '\r' { newline(); }
                | '\n' { newline(); }
                |  ~( '\n' | '\r' )
                )
            )*
            "*/"
        | '/' (( ~'\n'))* '\n' { newline(); }
        )
        { $setType(Token.SKIP); }
    ;


//Addapdted from antlr example java.g
// Whitespace -- ignored
WHITESPACE
     :   (  ' '
      |  '\t'
      |  '\f'
         // handle newlines
      |  (  options {generateAmbigWarnings=false;}
         :  "\r\n"  // Evil DOS
```

```
         |  '\r'    // Macintosh
         |  '\n'    // Unix (the right way)
         )
         { newline(); }
      )+
      { _ttype = Token.SKIP; }
   ;

// Single-line comments
SINGLE_COMMENT
   :  "//"
      (~('\n'|'\r'))* ('\n'|'\r'('\n')?)?
      {$setType(Token.SKIP); newline();}
   ;

// multiple-line comments
ML_COMMENT
   :  "/*"
      (  /* '\r' '\n' can be matched in one alternative or by matching
            '\r' in one iteration and '\n' in another.  I am trying to
            handle any flavor of newline that comes in, but the language
            that allows both "\r\n" and "\r" and "\n" to all be valid
            newline is ambiguous.  Consequently, the resulting grammar
            must be ambiguous.  I'm shutting this warning off.
         */
         options {
            generateAmbigWarnings=false;
         }
      :
         { LA(2)!='/' }? '*'
      |  '\r' '\n'      {newline();}
      |  '\r'        {newline();}
      |  '\n'        {newline();}
      |  ~('*'|'\n'|'\r')
      )*
      "*/"
      {$setType(Token.SKIP);}
   ;
```

```
class WclParser extends Parser;

options
{
    k = 2;
    buildAST = true;
    exportVocab = Wcl;
    ASTLabelType = "WclAST";
    defaultErrorHandler = false;
}

tokens {
    PROG;
    STATEMENT;
    EXPRESSION;
    CONDITION;
    THEN;
    NULL;
    INDEX;
    DECLARE;
    CALL_FUNCTION;
    FUNCTION;
    VARIABLE;
    GLOBAL;
    TYPE;
    ARGS;
    FUNCTION_BODY;
    OBJECT_METHOD;
    VALUE;
    CONSTRUCTOR;
    RETURN_VAL;

    NUM_INT;
    NUM_FLOAT;
}

program {
   getASTFactory().setASTNodeType("WclAST");
}
    : (globalStmt)* (functionDeclareStmt)* ( statement )* EOF!
```

```
        {#program = #([PROG, "PROG"], program); }
    ;

globalStmt
    : ("global"! | "declare") declareStmt
       {#globalStmt = #([GLOBAL, "GLOBAL"], globalStmt); }
    ;

functionDeclareStmt
    : "declare"! functionDeclare SEMICOLON!
       {#functionDeclareStmt = #([DECLARE, "DECLARE"], functionDeclareStmt); }
    ;

statement
    : whileStmt
    | breakStmt
    | continueStmt
    | ifStmt
    | functionStmt
    | declareStmt
    | returnStmt
    | functionCallStmt
    | objectReferenceStmt
    | evaluateStmt
    | LEFTBRACE! (statement)* RIGHTBRACE!
    ;

function_statement
    : whileStmt
    | breakStmt
    | continueStmt
    | ifStmt
    | declareStmt
    | returnStmt
    | functionCallStmt
    | objectReferenceStmt
    | evaluateStmt
    | LEFTBRACE! (statement)* RIGHTBRACE!
    ;
```

```
whileStmt
    : "while"^ expressionStmt statement
    ;

breakStmt : "break"^ SEMICOLON!;

continueStmt : "continue"^ SEMICOLON!;

returnStmt : "return"! expression SEMICOLON!
               {#returnStmt = #([RETURN_VAL, "RETURN_VAL"], returnStmt); }
           ;

ifStmt
    : "if"^ expressionStmt ifStmtBody elseStmt
    ;

ifStmtBody
    : statement
        {#ifStmtBody = #([THEN, "THEN"], ifStmtBody); }
    ;

expressionStmt
    : LEFTPAREN! expression RIGHTPAREN!
        {#expressionStmt = #([CONDITION, "CONDITION"], expressionStmt); }
    ;

elseStmt
    : (options {greedy = true;}
        : "else"^ statement
        )?
    ;

expression
    : compare (OR^ compare)*
    ;

compare
    : inverse (AND^ inverse)*
    ;
```

```
inverse
    : (NOT^)?
        compareTo
    ;

compareTo
    : eval ( ( LESS^ | LESSEQUAL^ | MORE^ | MOREEQUAL^
            | EQUALTO^ | NOTEQUAL^) eval ) *
    ;

eval
    : evalTerm ( ( PLUS^ | MINUS^)  evalTerm ) *
    ;

evalTerm
    : atom ( ( TIMES^ | SLASH^ | UNARY^ | MODULO^) atom) *
    | NOT atom
    | LEFTBRACKET! atom RIGHTBRACKET!
    ;

 atom
    : idValue
    | callFunction
    | numberValue
    | String
    | LEFTPAREN! expression RIGHTPAREN!
    ;

numberValue
    : NUM_INT
    | NUM_FLOAT
    ;

 primitives
    : "bool"
    | "int"
    | "float"
    | "string"
    | "WebPage"
    | "Link"
```

```
    | "Paragraph"
    | "Table"
    | "Image"
    ;

functionDeclare
    :
        functionDeclaration
        {#functionDeclare = #([FUNCTION, "FUNCTION"], functionDeclare); }
    ;

functionDeclaration
    :
        "function" idType idValue declareType
    ;

functionStmt
    :
        functionDeclaration functionBody
        {#functionStmt = #([FUNCTION, "FUNCTION"], functionStmt); }
    ;

 declareStmt
     :
        idType idValue declareType
        {#declareStmt = #([VARIABLE, "VARIABLE"], declareStmt); }
     ;

 idType
    : primitives
        {#idType = #([TYPE, "TYPE"], idType); }
    ;

declareType
    : LEFTPAREN! argDeclarationList RIGHTPAREN!
    | ( assignValue)?
        (","! ID ( assignValue)?)*
        SEMICOLON!
    ;
```

```
assignValue
    :  EQUAL! constructorStmt
    |  EQUAL! expression
        {#assignValue = #([VALUE, "VALUE"], assignValue); }
    ;

functionBody
    : function_statement
        {#functionBody = #([FUNCTION_BODY, "FUNCTION_BODY"], functionBody); }
    ;

constructorStmt
    : primitives argList
        {#constructorStmt = #([CONSTRUCTOR, "CONSTRUCTOR"], constructorStmt); }
    ;

argDeclarationList
    : (idType idValue
            (COMMA! idType idValue)*
        )?
      {#argDeclarationList = #([ARGS, "ARGS"], argDeclarationList); }
    ;

argList
    : LEFTPAREN! ((expression) (COMMA! expression)*)? RIGHTPAREN!
        {#argList = #([ARGS, "ARGS"], argList); }
    ;

evaluateStmt
    : idValue EQUAL expression SEMICOLON!
        {#evaluateStmt = #([STATEMENT, "STATEMENT"], evaluateStmt); }
    ;

idValue
    : ID index
    ;

index
    : LEFTBRACKET! indexValue RIGHTBRACKET! indexTail
        {#index = #([INDEX, "INDEX"], index); }
```

```
    | /* nothing */
    ;

indexTail
    : (LEFTBRACKET! indexValue RIGHTBRACKET!)?
    ;

indexValue
    : expression
    | /* empty */
        {#indexValue = #([NULL, "NULL"], indexValue); }
    ;

functionCallStmt
    : callFunction SEMICOLON!
    ;

callFunction
    : ID argList
        {#callFunction = #([CALL_FUNCTION, "CALL_FUNCTION"], callFunction); }
    ;

objectReferenceStmt
    : ID PERIOD! ID argList SEMICOLON!
        {#objectReferenceStmt = #([OBJECT_METHOD, "OBJECT_METHOD"], objectReferenceStmt)
    ;

evaluate_operators
    : EQUAL
    ;
```

A.2 WCL Semantics

```
{
   import java.util.HashSet;
   import java.util.Vector;
}

/**
 * The semantic check tree walker..
```

```
 *
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */
class WclSemanticCheck extends TreeParser;


options{
   importVocab = Wcl;
   ASTLabelType = "WclAST";
}

{
   /**
    * Global Variables
    */
   String prog = "WclSemanticCheck";
   WclSymbolTableStack sts = new WclSymbolTableStack();
   WclSymbolTable st;

   HashSet declaredFuncs = new HashSet();
   HashSet definedFuncs  = new HashSet();

   WclAST m,n;
   WclData arrayData;
   int line, arrayNumIndicesCalled=-1;
   boolean isFunction=false,isArrayAssignment=false,
           isOperator=false,isTableMethod=false;


   /**
    * Helper function: Counts number of indices in an array
    */
   public int getIndexCount(WclAST index) {
      int count=0;
      WclAST tmp = (WclAST)(index.getFirstChild());
      /* Loop through each index, skipping over INDEX nodes */
      while (tmp != null) {
         if (tmp.getType() != WclSemanticCheckTokenTypes.INDEX) {
            count++;
         }
```

```
      tmp = (WclAST)(tmp.getNextSibling());
   }
   return count;
}

/**
 * Helper function: Counts number of null indices in an array
 */
public int getNullIndexCount(WclAST index) {
   int count=0;
   WclAST tmp = (WclAST)(index.getFirstChild());
   /* Loop through each index, skipping over INDEX nodes */
   while (tmp != null) {
      if (tmp.getType() != WclSemanticCheckTokenTypes.INDEX &&
          tmp.getText() == WclData.getNull()) {
         count++;
      }
      tmp = (WclAST)(tmp.getNextSibling());
   }
   return count;
}

/**
 * Helper function: Verifies that an array has proper number of indices
 */
public int verifyIndexCount(String id, int lineNum, WclAST index) {
   WclData data = sts.getWclData(id);
   if (!data.isArray()) {
      Main.error(prog,lineNum,
                 "Identifier `" + id + "' is not an array.  " +
                 "Don't try to access it via an index.");
   }
   int calledCount = getIndexCount(index);
   int declaredCount = sts.getWclData(id).getNumIndices();
   if (calledCount != declaredCount) {
      Main.error(prog, lineNum,
                 "Array '" + id + "' was declared using " +
                 declaredCount + " indices, but you reference " +
                 calledCount + " indices.  You should fix that.");
   }
```

```
   return calledCount;
}

/**
 * Helper function: Verifies each array index is of type int
 */
public void verifyIndexType(String id, int lineNum, WclAST index) {
   WclAST tmp = (WclAST)(index.getFirstChild());
   String type="",tmpId="";
   WclData data=null;
   while (tmp != null) {
      tmpId = tmp.getText();
      /* Lookup type for identifiers */
      if (tmp.getType() == WclSemanticCheckTokenTypes.ID) {
         data = sts.getWclData(tmpId);

         /* First check if this is actually a bool */
         if (tmpId.equals("true") || tmpId.equals("false") ) {
            Main.error(prog,lineNum,
                 "Array indices can only be type `" + WclData.getInt()  +
                 "', but you're referencing type `" + WclData.getBool() + "'.");
         }

         /* identifier not declared */
         if (data == null) {
            Main.error(prog, lineNum,
                  "Cannot find identifier `" + tmpId + "'.");
         /* verify embedded arrays */
         } else if (data.isArray()) {
            /* Make sure there actually is an index */
            tmp = (WclAST)(tmp.getNextSibling());
            if (tmp == null) {
               Main.error(prog,lineNum,
                    "Unindexed array identifiers can only be refernced " +
                    "in an array variable assignment context.");
            /* Recursively check nested arrays */
            } else {

               /* Error if null index.
                * Embedded array indices imply that we need an actual value.
```

```
          */
         if (getNullIndexCount(tmp) != 0) {
            Main.error(prog,lineNum,"Must explicitly reference each "+
               "array index when using an array as an index to " +
               "another array.");
         }
         verifyIndexCount(tmpId,lineNum,tmp);
         verifyIndexType(tmpId,lineNum,tmp);
      }
   }
   type = data.getType();
} else if (tmp.getType() == WclSemanticCheckTokenTypes.NUM_INT) {
   type = WclData.getInt();
} else if (tmp.getType() == WclSemanticCheckTokenTypes.NUM_FLOAT) {
   type = WclData.getFloat();
} else if (tmp.getType() == WclSemanticCheckTokenTypes.String) {
   type = WclData.getString();
} else if (tmp.getType() == WclSemanticCheckTokenTypes.NULL) {
   type = WclData.getNull();

/* Treat the index as an expr and return result */
} else {
   try {
      type = this.expr(tmp);
   } catch (Exception e) {
      System.out.println("Something really bad happened!");
      System.exit(-1);
   }
}

/* Finally, check that type is int or null
 * Allow null at top level for array declarations with assignment.
 * Enforce null restrictions as necessary elsewhere.
 */
if (!type.equals(WclData.getInt()) &&
    !type.equals(WclData.getNull())) {
   Main.error(prog, lineNum,
           "Array indices can only be type `" + WclData.getInt()  +
           "', but you're referencing type `" + type + "'.");
}
```

```
         tmp = (WclAST)(tmp.getNextSibling());
      }
   }

   /**
    * Helper function: Counts number of paramaters in function dec/def
    */
   public int getParamCount(WclAST paramList) {
      WclAST m=null;
      WclAST n=null;
      int count = 0;
      m = (WclAST)(paramList.getFirstChild());
      while (m != null) {
         n = (WclAST)(m.getNextSibling().getNextSibling());
         if (n != null && n.getType() == WclSemanticCheck.INDEX) {
            n = (WclAST)n.getNextSibling();
         }
         count++;
         m = n;
      }
      return count;
   }

   /**
    * Helper function: For clarity below
    */
   public int getArgCount(WclAST arglist) {
      return getIndexCount(arglist);
   }

}


/**
 * Starts the walker checking semantics
 */
checkSemantics
   : #(PROG (stmt)*)
   {
      /* Make sure we defined every function declared */
```

```
      if (!declaredFuncs.isEmpty()) {
         String error = "Function ";
         if (declaredFuncs.size() > 1) {
            error = "Functions ";
         }
         Object[] leftovers = declaredFuncs.toArray();
         boolean flag = false;
         for (int i=0;i<leftovers.length-1;i++) {
            error += "`" + (String)leftovers[i] + "', ";
            flag = true;
         }
         if (flag) {
            error += "and ";
         }
         error += "`" + (String)leftovers[leftovers.length-1] + "' " +
                  "declared but not defined.";
         Main.error(prog, 0, error);
      }
   }
   ;


/**
 * In Wcl, expressions return values.
 *
 * This function returns the type of the value.
 */

expr returns [String r]
{
   String a=null,b=null,c=null,id=null,type=null;
   r=null;
   WclData data=null;  //local var
   arrayData=null;     //global var
   arrayNumIndicesCalled=-1;
}
   : NUM_INT              { r = WclData.getInt();}
   | NUM_FLOAT            { r = WclData.getFloat();}
   | String               { r = WclData.getString();}
   | NULL                 { r = WclData.getNull();}
```

```
| #(TYPE n:primitives) { r = n.getText(); }
| #(VALUE b=expr)      { r = b; }
| ID (l:INDEX)?
{
   /* First check if this is actually a bool */
   String truth = #ID.getText();
   if (truth.equals("true") ||
       truth.equals("false") ) {
       r = WclData.getBool();
       break;
   }

   /* Make sure id is in scope */
   id=#ID.getText();
   if (isFunction) { data=sts.getWclDataInFunction(id); }
   else { data=sts.getWclData(id); }
   if (data == null) {
      Main.error(prog, #ID.getLineNumber(),
                 "Cannot find identifier `" + id + "'.");
   }

   /* Extra processing for arrays */
   int calledCount = 0;
   if (l != null) {
      /* Verify ID is actually an array */
      if (!sts.getWclData(id).isArray()) {
         Main.error(prog, #ID.getLineNumber(),
                    "Identifier `" + id + "' is not an array.  " +
                    "Don't try to access it via an index.");
      }

      /* Verify index count */
      calledCount = verifyIndexCount(id,#ID.getLineNumber(),#l);

      /* Verify each index is type int */
      verifyIndexType(id,#ID.getLineNumber(),#l);

   /* Ok, no array index, still more array checks */
   } else if (data.isArray()) {
```

```
      /* Arrays can be referenced w/out indexing in limited context */
      if (isArrayAssignment) {

         /* Cannot use binary operators on unindexed arrays */
         if (isOperator) {
            Main.error(prog, #ID.getLineNumber(),
                       "Cannot use binary operators on unindexed arrays.");

         /* Set global vars to handle array assignment */
         } else {
            arrayNumIndicesCalled=calledCount;
            arrayData=data;
         }

      /* Need to save array info for Table.add method check later */
      } else if (isTableMethod) {
         arrayNumIndicesCalled=calledCount;
         arrayData=data;

      /* Must reference array identifiers using indeces in other contexts */
      } else {
         Main.error(prog, #ID.getLineNumber(),
                    "Unindexed array identifiers can only be refernced " +
                    "in an array variable assignment context.");
      }
   }

   /* Finally return type */
   r = data.getType();
}
| #(PLUS {isOperator=true;} a=expr b=expr {isOperator=false;})
{
   type = "ERROR";

   /* Plus is overloaded in Wcl, so if one of the operators is a string
    * or an object, then make then use string concatentation.
    */
   if (WclData.isString(a) || WclData.isString(b) ||
       WclData.isWebPage(a) || WclData.isParagraph(a) ||
       WclData.isLink(a) || WclData.isImage(a) || WclData.isTable(a) ||
```

```
        WclData.isWebPage(b) || WclData.isParagraph(b) ||
        WclData.isLink(b) || WclData.isImage(b) || WclData.isTable(b) )
   {
      type = WclData.getString();

   /* Otherwise use binary addition and types must be the same */
   } else if (!a.equals(b) ) {
      Main.error(prog, #PLUS.getLineNumber(),
                 "Operators only work on same type.  " +
                 "You're mixing `" + a + "' and `" + b + "'.");

   /* If we're here, then it's binary addition and types are same,
    * so return one of them.
    */
   } else {
      type = a;
   }

   r = type;
}
| #(MINUS {isOperator=true;} a=expr b=expr {isOperator=false;})
{
   /* MINUS operator only works on floats or ints */
   if (!WclData.isInt(a) && !WclData.isFloat(a) ) {
      Main.error(prog, #MINUS.getLineNumber(),
                 "Subtraction operator only works on types `" +
                 WclData.getInt() + "' or `" + WclData.getFloat() +
                 "'.  You're trying to use type `" + a + "'.");
   }
   if (!WclData.isInt(b) && !WclData.isFloat(b) ) {
      Main.error(prog, #MINUS.getLineNumber(),
                 "Subtraction operator only works on types `" +
                 WclData.getInt() + "' or `" + WclData.getFloat() +
                 "'.  You're trying to use type `" + b + "'.");
   }

   /* Operators only work on same type */
   if (!a.equals(b)) {
      Main.error(prog, #MINUS.getLineNumber(),
                 "Operators only work on same type.  " +
```

```
                "You're mixing `" + a + "' and `" + b + "'.");
   }
   r = a;        // Otherwise types are same, so return one of them.
}
| #(TIMES {isOperator=true;} a=expr b=expr {isOperator=false;})
{
   /* TIMES operator only works on floats or ints */
   if (!WclData.isInt(a) && !WclData.isFloat(a) ) {
      Main.error(prog, #TIMES.getLineNumber(),
                "Multiplication operator only works on types `" +
                WclData.getInt() + "' or `" + WclData.getFloat() +
                "'.  You're trying to use type `" + a + "'.");
   }
   if (!WclData.isInt(b) && !WclData.isFloat(b) ) {
      Main.error(prog, #TIMES.getLineNumber(),
                "Multiplication operator only works on types `" +
                WclData.getInt() + "' or `" + WclData.getFloat() +
                "'.  You're trying to use type `" + b + "'.");
   }

   /* Operators only work on same type */
   if (!a.equals(b)) {
      Main.error(prog, #TIMES.getLineNumber(),
                "Operators only work on same type.  " +
                "You're mixing `" + a + "' and `" + b + "'.");
   }
   r = a;        // Otherwise types are same, so return one of them.
}
| #(SLASH {isOperator=true;} a=expr b=expr {isOperator=false;})
{
   /* SLASH operator only works on floats or ints */
   if (!WclData.isInt(a) && !WclData.isFloat(a) ) {
      Main.error(prog, #SLASH.getLineNumber(),
                "Division operator only works on types `" +
                WclData.getInt() + "' or `" + WclData.getFloat() +
                "'.  You're trying to use type `" + a + "'.");
   }
   if (!WclData.isInt(b) && !WclData.isFloat(b) ) {
      Main.error(prog, #SLASH.getLineNumber(),
                "Division operator only works on types `" +
```

```
                WclData.getInt() + "' or `" + WclData.getFloat() +
                "'.  You're trying to use type `" + b + "'.");
   }

   /* Operators only work on same type */
   if (!a.equals(b)) {
      Main.error(prog, #SLASH.getLineNumber(),
                "Operators only work on same type.  " +
                "You're mixing `" + a + "' and `" + b + "'.");
   }
   r = a;        // Otherwise types are same, so return one of them.
}
| #(MODULO {isOperator=true;} a=expr b=expr {isOperator=false;})
{
   /* Modulo only works on integers */
   if (!WclData.isInt(a)) {
      Main.error(prog, #MODULO.getLineNumber(),
                "Modulo only work on int.  " +
                "You're using `" + a + "`");
   }
   if (!WclData.isInt(b)) {
      Main.error(prog, #MODULO.getLineNumber(),
                "Modulo only work on int.  " +
                "You're using `" + b + "`");
   }
   r = a;        // Otherwise types are same, so return one of them.
}
| #(CALL_FUNCTION ID ARGS)
{
   /* Make sure ID is declared and is for a function */
   id=#ID.getText();
   data=sts.getWclData(id);
   if (data == null || !data.isFunction()) {
      Main.error(prog, #ID.getLineNumber(),
                "Cannot find function `" + id + "'.");
   }

   /* Make sure # of args match declaration */
   Vector v = data.getArgs();
   int count = getArgCount(#ARGS);
```

```
if (count != v.size()) {
   String arg1 = "arguments";
   String arg2 = "arguments";
   if (v.size() == 1) { arg1 = "argument"; }
   if (  count  == 1) { arg2 = "argument"; }
      Main.error(prog, #ID.getLineNumber(), "Expecting " + v.size() +
              " " + arg1 + ", but found " + count + " " + arg2 + ".");
}

/* Make sure each arg type matches parameter declaration */
int i = 0;
type = "";
int calledCount = -1;  // for safety
int nullCount   = -1;  // for safety
WclData argData = null;
WclFunctionArg paramData = null;
String declaredType = "";
m = (WclAST)(#ARGS.getFirstChild());
while (m != null) {
   paramData = (WclFunctionArg)v.elementAt(i);
   declaredType = ((WclFunctionArg)v.elementAt(i)).getType();

   /* Peek ahead to the next argument */
   n = (WclAST)(m.getNextSibling());

   /* Lookup the type of the argument */
   if (m.getType() == WclSemanticCheckTokenTypes.ID) {

      /* Identifier not declared */
      argData = sts.getWclData(m.getText());
      if (argData == null) {
         Main.error(prog, #ID.getLineNumber(),
                  "Cannot find identifier `" + m.getText() + "'.");
      }

      /* Verify indexed arrays */
      if (n != null && n.getType() == WclSemanticCheckTokenTypes.INDEX) {
         /* Make sure no index is null */
         nullCount = getNullIndexCount(n);
         if (nullCount != 0) {
```

```
            Main.error(prog, #ID.getLineNumber(), "Arrays arguments to "+
               "function calls must explicitly reference each index.  " +
               "If you want to pass the entire array, just use the " +
               "identifier and omit the brackets [].");
         }

         calledCount = verifyIndexCount(m.getText(),#ID.getLineNumber(),n);
         verifyIndexType(m.getText(),#ID.getLineNumber(),n);
      }
      type = argData.getType();
      } else if (m.getType() == WclSemanticCheckTokenTypes.NUM_INT) {
         type = WclData.getInt();
      } else if (m.getType() == WclSemanticCheckTokenTypes.NUM_FLOAT) {
         type = WclData.getFloat();
      } else if (m.getType() == WclSemanticCheckTokenTypes.String) {
         type = WclData.getString();

      /* Treat the argument as an expr and return result */
      } else {
         try {
            type = this.expr(m);
         } catch (Exception e) {
            System.out.println("Something really bad happened!");
            System.exit(-1);
         }
      }

   /* Check if parameter was declared as array */
   if ( ((WclFunctionArg)v.elementAt(i)).getCategory() ==
          WclFunctionArg.ARRAY) {

      /* Arg not an array */
      if (argData == null || !argData.isArray()) {
         Main.error(prog,#ID.getLineNumber(),"Expecting an array but `" +
                    m.getText() + "' is type `" + type + "'.");
      }

      /* Arg references index */
      if (calledCount > 0) {
         Main.error(prog,#ID.getLineNumber(),"Expecting an array but `" +
```

```
                        m.getText() + "' references type `" + type + "'.  " +
                        "Try referencing the array without indexing.");
            }

            /* Arrays not compatible dimensions */
            if (argData.getNumIndices() != paramData.getDimensions()) {
               Main.error(prog,#ID.getLineNumber(),"Expecting an array of " +
                        paramData.getDimensions() + " dimensions, but `" +
                        m.getText() + "' is " + argData.getNumIndices() +
                        " dimensions.");
            }

         /* Else paramater not declared as array */
         } else {
            /* Make sure argument not an unindexed array */
            if (argData != null && argData.isArray() && n == null) {
               Main.error(prog,#ID.getLineNumber(),"Expecting an argument of "+
                        "type `" + declaredType + "', but you are passing an "+
                        "unindexed array.");
            }
         }

         /* Finally check the types */
         if (!type.equals(declaredType)) {
            Main.error(prog, #ID.getLineNumber(),
                     "Expecting type `" + declaredType  + "', but " +
                     "found type `" + type + "'.");
         }

         /* Iterate to next argument */
         m = (WclAST)(m.getNextSibling());
         if (n != null && n.getType() == WclSemanticCheckTokenTypes.INDEX) {
            m = (WclAST)(m.getNextSibling());
         }
         i++;
      }

      /* Finally return the function's type */
      r = data.getType();
   }
```

```
/*
 * Conditionals
 */
| #(AND a=expr b=expr)
{
   if (!WclData.isBool(a) || !WclData.isBool(b)) {
      Main.error(prog, #AND.getLineNumber(),
                 "AND operator only works on type `"+WclData.getBool()+"' "+
                 "not on types `"+a+"' and `"+b+"'.");
   }
   r = WclData.getBool();
}
| #(OR a=expr b=expr)
{
   if (!WclData.isBool(a) || !WclData.isBool(b)) {
      Main.error(prog, #OR.getLineNumber(),
                 "OR operator only works on type `"+WclData.getBool()+"' "+
                 "not on types `"+a+"' and `"+b+"'.");
   }
   r = WclData.getBool();
}
| #(LESS a=expr b=expr)
{
   if (!WclData.isInt(a) && !WclData.isFloat(a) ||
       !WclData.isInt(b) && !WclData.isFloat(b)) {
      Main.error(prog, #LESS.getLineNumber(),"< operator only works on " +
                  "types `"+WclData.getInt()+"' and `"+WclData.getFloat()+
                  "'.  You're using types `" + a + "' and `" + b + "'.");
   }
   r = WclData.getBool();
}
| #(MORE a=expr b=expr)
{
   if (!WclData.isInt(a) && !WclData.isFloat(a) ||
       !WclData.isInt(b) && !WclData.isFloat(b)) {
      Main.error(prog, #MORE.getLineNumber(),"> operator only works on " +
                  "types `"+WclData.getInt()+"' and `"+WclData.getFloat()+
                  "'.  You're using types `" + a + "' and `" + b + "'.");
   }
   r = WclData.getBool();
```

```
}
| #(EQUALTO a=expr b=expr)
{
   if (!a.equals(b)) {
      /* Mixing terms and they aren't int or float */
      if (!WclData.isInt(a) && !WclData.isFloat(a) ||
          !WclData.isInt(b) && !WclData.isFloat(b)) {
         Main.error(prog, #EQUALTO.getLineNumber(),"== operator only " +
                    "works on different types if those types are `" +
                    WclData.getInt()+"' and `" + WclData.getFloat()+
                    "'.  You're using types `" + a + "' and `" + b + "'.");
      }
   }
   r = WclData.getBool();
}
| #(NOTEQUAL a=expr b=expr)
{
   if (!a.equals(b)) {
      /* Mixing terms and they aren't int or float */
      if (!WclData.isInt(a) && !WclData.isFloat(a) ||
          !WclData.isInt(b) && !WclData.isFloat(b)) {
         Main.error(prog, #NOTEQUAL.getLineNumber(),"!= operator only " +
                    "works on different types if those types are `" +
                    WclData.getInt()+"' and `" + WclData.getFloat()+
                    "'.  You're using types `" + a + "' and `" + b + "'.");
      }
   }
   r = WclData.getBool();
}

| #(LESSEQUAL a=expr b=expr)
{
   if (!WclData.isInt(a) && !WclData.isFloat(a) ||
       !WclData.isInt(b) && !WclData.isFloat(b)) {
      Main.error(prog, #LESSEQUAL.getLineNumber(),"<= operator only works "+
                 "on types `"+WclData.getInt()+"' and `"+WclData.getFloat()+
                 "'.  You're using types `" + a + "' and `" + b + "'.");
   }
   r = WclData.getBool();
}
```

```
   | #(MOREEQUAL a=expr b=expr)
   {
      if (!WclData.isInt(a) && !WclData.isFloat(a) ||
          !WclData.isInt(b) && !WclData.isFloat(b)) {
         Main.error(prog, #MOREEQUAL.getLineNumber(),">= operator only works "+
                    "on types `"+WclData.getInt()+"' and `"+WclData.getFloat()+
                    "'.  You're using types `" + a + "' and `" + b + "'.");
      }
      r = WclData.getBool();
   }
   ;

stmt
{
   String a=null,b=null,c=null,id=null,type=null,index=null;
   WclData data=null;        //local var
   arrayData=null;           //global var
   arrayNumIndicesCalled=-1; //for good measure
}

   /*
    * While loop, need new scope
    */
   : #(w:"while" {sts.enterNewScope();} #(CONDITION a=expr)
      {
         if (!WclData.isBool(a)) { Main.error(prog,w.getLineNumber(),
               "Expression does not evaluate to type `"+WclData.getBool()+"'.");
         }
      }
      (stmt)* {sts.exitNewScope();})

   /*
    * Conditional if, then, else, need new scope
    */
   | #(f:"if" #(CONDITION a=expr)
      {
         if (!WclData.isBool(a)) { Main.error(prog,f.getLineNumber(),
               "Expression does not evaluate to type `"+WclData.getBool()+"'.");
         }
      }
```

```
  #(THEN {sts.enterNewScope();} (stmt)* {sts.exitNewScope();})
  (#("else" {sts.enterNewScope();} (stmt)* {sts.exitNewScope();}))?
   )

/*
 * Function declaration
 */
| #(DECLARE #(FUNCTION "function" a=expr ID ARGS) )
{
   /* Make sure identifier is not a reserved word */
   id = #ID.getText();
   if (WclReservedWords.isReserved(id)) {
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                  id + "' is a reserved word.  Pick a new name.");
   }

   /* Check if identifier already declared */
   if (sts.hasLocal(id)) {
      /* id already declared in this scope, so error */
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                  id + "' already declared.");
   }

   /* Grab each parameter */
   m = (WclAST)(#ARGS.getFirstChild());
   Vector v = new Vector();
   WclFunctionArg arg = null;
   int indexCount = -1;  // for good measure
   int nullCount  = -1;  // for good measure
   while (m != null) {
      /* Peek ahead to next parameter */
      n = (WclAST)(m.getNextSibling().getNextSibling());

      /* Parameter is an array */
      if (n != null && n.getType() == WclSemanticCheck.INDEX) {

         /* Make sure indicies are all null */
         nullCount = getNullIndexCount(n);
         indexCount = getIndexCount(n);
         if (nullCount != indexCount) {
```

```
            Main.error(prog, #ID.getLineNumber(), "Cannot explicitly " +
                  "specify size of array index in function parameter " +
                  "declaration.");
         }

         /* Otherwise record index */
         arg = new WclFunctionArg(m.getFirstChild().getText(),
                              WclFunctionArg.ARRAY, indexCount);
         n = (WclAST)n.getNextSibling();

      /* Parameter is a normal variable */
      } else {
         arg = new WclFunctionArg(m.getFirstChild().getText(),
                                    WclFunctionArg.VARIABLE);
      }

      /* Add parameter to vector, and move on to next parameter */
      v.add(arg);
      m = n;
   }

   /* Add function to symbol table */
   sts.addId(id,new WclData(a,WclData.getFunction(),v));
   /* Add function to set to make sure it is later defined */
   declaredFuncs.add(id);

}

/*
 * Function Definitions
 * Here we set isFunction=true so that the stmts matched in FUNCTION_BODY
 * so we can manage scoping more easily.
 */

| #(FUNCTION {isFunction=true;sts.enterNewScope();} "function" a=expr ID ARGS

/* Ok, need to add ARGS to symbol table now, otherwise the STMTS in function
 * body will be checked before the ARGS are in the symbol table.  So, do
 * some other checks while we're at it.
 */
```

```
{
   id = #ID.getText();
   data=sts.getWclData(id);

   /* Make sure function wasn't previously defined */
   if (definedFuncs.contains(id)) {
      Main.error(prog, #ID.getLineNumber(), "Function `" + id + "' " +
                 "previously defined.");
   }

   /* Make sure function was declared first */
   if (!declaredFuncs.contains(id)) {
      Main.error(prog, #ID.getLineNumber(), "Function `" + id + "' " +
                 "was not declared before trying to define it.");
   }

   String returnType = data.getType();

   /* Make sure # of args match declaration */
   Vector v = data.getArgs();
   int count = getParamCount(#ARGS);
   if (count != v.size()) {
      String arg1 = "parameters";
      String arg2 = "parameters";
      if (v.size() == 1) { arg1 = "parameter"; }
      if (  count  == 1) { arg2 = "parameter"; }
         Main.error(prog, #ID.getLineNumber(), "Expecting " + v.size() +
                 " " + arg1 + ", but found " + count + " " + arg2 + ".");
   }

   /* Make sure definition arguments' type and order matches declaration */
   /* Does not enforce consistent variable naming */
   int i = 0;
   type = "";
   WclFunctionArg declaredArg=null;
   m = (WclAST)(#ARGS.getFirstChild());
   int nullCount = -1; //for safety

   while (m != null) {
      n = (WclAST)(m.getNextSibling().getNextSibling());
```

```
declaredArg=(WclFunctionArg)v.elementAt(i);
type = m.getFirstChild().getText();

/* It's an array parameter */
if (n != null && n.getType() == WclSemanticCheck.INDEX) {

   /* Make sure this parameter was declared as an array */
   if (declaredArg.getCategory() != WclFunctionArg.ARRAY) {
      Main.error(prog, #ID.getLineNumber(),"Function parameter `" +
                 m.getNextSibling().getText() + "' was not " +
                 "declared as an array." );
   }

   /* Make sure param is referenced with empty indices */
   nullCount = getNullIndexCount(n);
   count = getIndexCount(n);
   if (nullCount != count) {
      Main.error(prog, #ID.getLineNumber(), "Cannot explicitly " +
            "specify size of array index in function parameter " +
            "definition.");
   }

   /* Make sure param was declared as array of same type */
   if (!type.equals(declaredArg.getType())) {
      Main.error(prog,#ID.getLineNumber(),"Function parameter `" +
                 m.getNextSibling().getText() + "' was declared as " +
                 "an array of `" + declaredArg.getType() + "', but " +
                 "you are defining it as an array of `" + type + "'.");
   }

   /* Make sure param was declared as an array of equal dimension */
   if (count != declaredArg.getDimensions()) {
      Main.error(prog,#ID.getLineNumber(),"Function parameter `" +
                 m.getNextSibling().getText() + "' was declared as " +
                 "an array with `" + count + "' indices, but you " +
                 "are defining it with `" + declaredArg.getDimensions() +
                 "' indices.");
   }

   /* Looks good, so add parameter to symbol table */
```

```
            String paramId = m.getNextSibling().getText();
            sts.addId(paramId,new WclData(type,WclData.getArray(),count));
            n = (WclAST)n.getNextSibling();

         /* It's a regular variable parameter */
         } else {

            /* Make sure this parameter was declared as variable */
            if (declaredArg.getCategory() != WclFunctionArg.VARIABLE) {
               Main.error(prog, #ID.getLineNumber(),"Function parameter `" +
                          m.getNextSibling().getText() + "' was not " +
                          "declared as a regular variable." );
            }

            /* Make sure param was declared as variable of same type */
            if (!type.equals(declaredArg.getType())) {
               Main.error(prog,#ID.getLineNumber(),"Function parameter `" +
                          m.getNextSibling().getText() + "' was declared as " +
                          "type `" + declaredArg.getType() + "', but you are "+
                          "defining it as type `" + type + "'.");
            }

            /* Looks good, so add parameter to symbol table */
            String paramId = m.getNextSibling().getText();
            sts.addId(paramId,new WclData(type,WclData.getVariable()));
         }

         /* On to the next parameter */
         m = n;
         i++;
      }


}

/*
 * Ok, now continue matching rest of function body
 */
   #(FUNCTION_BODY (options {greedy=true;} : stmt)* (#(RETURN_VAL b=expr))?)
   {isFunction=false;sts.exitNewScope();})
```

```
{
   /* Ensure that function returns a value as last statement */
   if (b == null) {
      Main.error(prog, #ID.getLineNumber(), "Function `" + id + "' " +
              "is supposed to return a value as the last statement of " +
              "the function.  This return statement is missing.");
   }

   /* Ensure that function returns proper type through return statement */
   if (!returnType.equals(b)) {
      Main.error(prog, #ID.getLineNumber(), "Function `" + id + "' " +
              "is supposed to return a value of type `" + returnType +
              "'.  However, you are returning a value of type `" + b +
              "'.");
   }

   /* Finally, mark that we defined the function */
   definedFuncs.add(id);

   /* Remove from declaredFuncs set */
   declaredFuncs.remove(id);
}


/*
 * Function call
 */
| #(CALL_FUNCTION ID ARGS)
{
   /* Make sure ID is declared and is for a function */
   id=#ID.getText();
   data=sts.getWclData(id);
   if (data == null || !data.isFunction()) {
      Main.error(prog, #ID.getLineNumber(),
                 "Cannot find function `" + id + "'.");
   }

   /* Make sure # of args match declaration */
   Vector v = data.getArgs();
```

```
int count = getArgCount(#ARGS);
if (count != v.size()) {
   String arg1 = "arguments";
   String arg2 = "arguments";
   if (v.size() == 1) { arg1 = "argument"; }
   if (  count  == 1) { arg2 = "argument"; }
      Main.error(prog, #ID.getLineNumber(), "Expecting " + v.size() +
              " " + arg1 + ", but found " + count + " " + arg2 + ".");
}

/* Make sure each arg type matches parameter declaration */
int i = 0;
type = "";
int calledCount = -1;  // for safety
int nullCount   = -1;  // for safety
WclData argData = null;
WclFunctionArg paramData = null;
String declaredType = "";
m = (WclAST)(#ARGS.getFirstChild());
while (m != null) {
   paramData = (WclFunctionArg)v.elementAt(i);
   declaredType = ((WclFunctionArg)v.elementAt(i)).getType();

   /* Peek ahead to the next argument */
   n = (WclAST)(m.getNextSibling());

   /* Lookup the type of the argument */
   if (m.getType() == WclSemanticCheckTokenTypes.ID) {

      /* Identifier not declared vs boolean*/
      argData = sts.getWclData(m.getText());
      if (argData == null && !m.getText().equals("true") &&
                             !m.getText().equals("false")) {
         Main.error(prog, #ID.getLineNumber(),
               "Cannot find identifier `" + m.getText() + "'.");
      }

      /* Verify indexed arrays */
      if (n != null && n.getType() == WclSemanticCheckTokenTypes.INDEX) {
```

```
      /* Make sure no index is null */
      nullCount = getNullIndexCount(n);
      if (nullCount != 0) {
         Main.error(prog, #ID.getLineNumber(), "Arrays arguments to "+
            "function calls must explicitly reference each index.  " +
            "If you want to pass the entire array, just use the " +
            "identifier and omit the brackets [].");
      }

      calledCount = verifyIndexCount(m.getText(),#ID.getLineNumber(),n);
      verifyIndexType(m.getText(),#ID.getLineNumber(),n);
   }

   /* Quick fix for grammer bug, not time to make better */
   if (m.getText().equals("true") || m.getText().equals("false")) {
      type = WclData.getBool();
   } else {
      type = argData.getType();
   }

} else if (m.getType() == WclSemanticCheckTokenTypes.NUM_INT) {
   type = WclData.getInt();
} else if (m.getType() == WclSemanticCheckTokenTypes.NUM_FLOAT) {
   type = WclData.getFloat();
} else if (m.getType() == WclSemanticCheckTokenTypes.String) {
   type = WclData.getString();

/* Treat the argument as an expr and return result */
} else {
   try {
      type = this.expr(m);
   } catch (Exception e) {
      System.out.println("Something really bad happened!");
      System.exit(-1);
   }
}

/* Check if parameter was declared as array */
if ( ((WclFunctionArg)v.elementAt(i)).getCategory() ==
       WclFunctionArg.ARRAY) {
```

```
   /* Arg not an array */
   if (argData == null || !argData.isArray()) {
      Main.error(prog,#ID.getLineNumber(),"Expecting an array but `" +
                 m.getText() + "' is type `" + type + "'.");
   }

   /* Arg references index */
   if (calledCount > 0) {
      Main.error(prog,#ID.getLineNumber(),"Expecting an array but `" +
                 m.getText() + "' references type `" + type + "'.  " +
                 "Try referencing the array without indexing.");
   }

   /* Arrays not compatible dimensions */
   if (argData.getNumIndices() != paramData.getDimensions()) {
      Main.error(prog,#ID.getLineNumber(),"Expecting an array of " +
                 paramData.getDimensions() + " dimensions, but `" +
                 m.getText() + "' is " + argData.getNumIndices() +
                 " dimensions.");
   }

/* Else paramater not declared as array */
} else {
   /* Make sure argument not an unindexed array */
   if (argData != null && argData.isArray() && n == null) {
      Main.error(prog,#ID.getLineNumber(),"Expecting an argument of "+
                "type `" + declaredType + "', but you are passing an "+
                "unindexed array.");
   }
}

/* Finally check the types */
if (!type.equals(declaredType)) {
   Main.error(prog, #ID.getLineNumber(),
           "Expecting type `" + declaredType  + "', but " +
           "found type `" + type + "'.");
}

/* Iterate to next argument */
```

```
      m = (WclAST)(m.getNextSibling());
      if (n != null && n.getType() == WclSemanticCheckTokenTypes.INDEX) {
         m = (WclAST)(m.getNextSibling());
      }
      i++;
   }

}

/*
 * Variable declaration with optional assignment
 */
| #(VARIABLE a=expr ID (y:INDEX)? {if (y != null) {isArrayAssignment=true;}}
   (b=expr)? {isArrayAssignment=false;})
{
   /* Make sure identifier is not a reserved word */
   id = #ID.getText();
   if (WclReservedWords.isReserved(id)) {
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                  id + "' is a reserved word.  Pick a new name.");
   }

   /* Make sure not already declared */
   if (sts.hasLocal(id)) {
      // id already declared in this scope, so error
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                  id + "' already declared in this scope.");
   }

   /* Add array to symbol table */
   if (y != null) {
      /* Find num dimensions */
      int indexCount = getIndexCount(#y);
      int nullCount = getNullIndexCount(#y);

      /* Add to symbol table now, break on error later */
      sts.addId(id,new WclData(a,WclData.getArray(),indexCount));

      /* Verify that each index is valid value */
      verifyIndexType(id,#ID.getLine(),#y);
```

```
/* This is an array assignment */
if (b != null) {

   /* Make sure array does not reference an index */
   if (nullCount != indexCount) {
      Main.error(prog, #ID.getLineNumber(), "Cannot reference an " +
                                "array index in array assignment.");
   }

   /* Make sure the expr evaluates to an array */
   if (arrayData == null || !arrayData.isArray() ||
       arrayNumIndicesCalled != 0) {
      Main.error(prog, #ID.getLineNumber(), "Assignment must be to " +
              "an unindexed array in this array assignment context.");
   }

   /* Make sure expr is an array of same type */
   if (!a.equals(arrayData.getType())) {
      Main.error(prog, #ID.getLineNumber(), "Array types not " +
                 "compatible.  You're mixing array types `" + a+
                 "' and `" + arrayData.getType() + "'.");
   }

   /* Make sure expr is an array of same dimensions */
   if (indexCount != arrayData.getNumIndices()) {
      Main.error(prog, #ID.getLineNumber(), "Array dimensions " +
           "not compatible.  You're mixing arrays of `" + indexCount +
           "' and `" + arrayData.getNumIndices() + "' dimension(s).");
   }

/* There is no assignment, but there is an index */
} else {
   /* Array in this context must reference each indice */
   if (nullCount != 0) {
      Main.error(prog, #ID.getLineNumber(), "An array variable " +
            "declaration without an assignment must explicitly " +
            "reference each indice.");
   }
}
```

```
   /* Add regular variable to symbol table */
   } else {
      /* Make sure expression evaluates to type appropriate for id */
      if (b != null && !a.equals(b)) {
         Main.error(prog, #ID.getLineNumber(), "Looking for `" +
                    a + "', but found `" + b +"'.");
      }

      /* Regular variable, add to symbol table */
      sts.addId(id,new WclData(a,WclData.getVariable()));
   }

}

/*
 * Global variable declaration with optional assignment
 */
| #(GLOBAL #(VARIABLE a=expr ID (z:INDEX)? {if (z != null) {isArrayAssignment=true;}}
   (b=expr)?) {isArrayAssignment=false;})
{

   /* Make sure identifier is not a reserved word */
   id = #ID.getText();
   if (WclReservedWords.isReserved(id)) {
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                 id + "' is a reserved word.  Pick a new name.");
   }

   /* Make sure not already declared */
   if (sts.hasGlobal(id)) {
      /* id already declared in global scope, so error */
      Main.error(prog, #ID.getLineNumber(), "Identifier `" +
                 id + "' already declared globally.");
   }

   /* Add array to symbol table */
   if (z != null) {

      /* Find num dimensions */
```

```
int indexCount = getIndexCount(#z);
int nullCount = getNullIndexCount(#z);

/* Add to symbol table now, break on error later */
sts.addGlobal(id,new WclData(a,WclData.getArray(),indexCount));

/* Verify that each index is valid value */
verifyIndexType(id,#ID.getLine(),#z);

/* This is an array assignment */
if (b != null) {

   /* Make sure array does not reference an index */
   if (nullCount != indexCount) {
      Main.error(prog, #ID.getLineNumber(), "Cannot reference an " +
                                "array index in array assignment.");
   }

   /* Make sure the expr evaluates to an array */
   if (arrayData == null) {
      Main.error(prog, #ID.getLineNumber(), "Assignment must be to " +
              "an unindexed array in this array assignment context.");
   }

   /* Make sure expr is an array of same type */
   if (!a.equals(arrayData.getType())) {
      Main.error(prog, #ID.getLineNumber(), "Array types not " +
                 "compatible.  You're mixing array types `" + a+
                 "' and `" + arrayData.getType() + "'.");
   }

   /* Make sure expr is an array of same dimensions */
   if (indexCount != arrayData.getNumIndices()) {
      Main.error(prog, #ID.getLineNumber(), "Array dimensions " +
           "not compatible.  You're mixing arrays of `" + indexCount +
           "' and `" + arrayData.getNumIndices() + "' dimension(s).");
   }

/* There is no assignment, but there is an index */
} else {
```

```
         /* Array in this context must reference each indice */
         if (nullCount != 0) {
            Main.error(prog, #ID.getLineNumber(), "An array variable " +
                  "declaration without an assignment must explicitly " +
                  "reference each indice.");
         }
      }

   /* Add regular variable to symbol table */
   } else {

      /* Make sure expression evaluates to type appropriate for id */
      if (b != null && !a.equals(b)) {
         Main.error(prog, #ID.getLineNumber(), "Looking for `" +
                    a + "', but found `" + b +"'.");
      }
      /* Regular variable, add to symbol table */
      sts.addGlobal(id,new WclData(a,WclData.getVariable()));
   }

}

/*
 * Variable assignment
 *
 * This is broken across two sections because we need to flag array
 * assignment before we match the expr.
 */
| #(STATEMENT ID (x:INDEX)? EQUAL
   {
      id = #ID.getText( );
      data = sts.getWclData(id);

      /* First check id is in scope */
      if (data == null) {
         Main.error(prog, #ID.getLineNumber(), "Variable `" +
                    id + "' not in any visible scope.");
      }

      /* Make sure id is not a function */
```

```
      if (data.isFunction()) {
         Main.error(prog, #ID.getLineNumber(), "Cannot assign a value " +
                    "to a function identifier.");
      }

      /* Flag that we're doing array assignment, either way */
      if (data.isArray()) {
         isArrayAssignment=true;
      }
   }
/*
 * Continue matching variable assignment from above
 */
   a=expr {isArrayAssignment=false;})
{
   /* Processing for arrays */
   if (x != null) {
      /* Verify ID is actually an array */
      if (!sts.getWclData(id).isArray()) {
         Main.error(prog, #ID.getLineNumber(),
                    "Identifier `" + id + "' is not an array.  " +
                    "Don't try to access it via an index.");
      }

      /* Since we're assigning a value to an array that is indexed,
       * make sure that none of the indices is null */
      int nullCount = getNullIndexCount(#x);
      if (nullCount != 0) {
         Main.error(prog, #ID.getLineNumber(), "An array variable " +
               "declaration without an assignment must explicitly " +
               "reference each indice.");
      }

      /* Verify index count */
      int calledCount = verifyIndexCount(id,#ID.getLineNumber(),#x);

      /* Verify each index is type int */
      verifyIndexType(id,#ID.getLineNumber(),#x);

      /* Now check that expr is valid for array element */
```

```
      if (!a.equals(data.getType())) {
         Main.error(prog, #ID.getLineNumber(), "Looking for `" +
                    data.getType() + "', but found `" + a +"'.");
      }

   /* Processing for non-indexed variables,
    * i.e. normal variables and array identifiers.
    * Make sure expression evaluates to type appropriate for id
    */
   } else {

      /* Array identifier, no index */
      if (data.isArray()) {

         /* Make sure array is assigned a compatible unindexed array */
         if (arrayNumIndicesCalled != 0 ||
              data.getNumIndices() != arrayData.getNumIndices() ) {
            Main.error(prog, #ID.getLineNumber(), "Only an unindexed " +
                       "array may be assigned to an unindexed array, " +
                       "and both arrays must be of equal dimension.");
         }

         /* Make sure expr is an array of same type */
         if (!data.getType().equals(arrayData.getType())) {
            Main.error(prog, #ID.getLineNumber(), "Array types not " +
               "compatible.  You're mixing array types `" + data.getType() +
               "' and `" + arrayData.getType() + "'.");
         }

      /* Regular identifier */
      } else if  (!a.equals(data.getType())) {
         Main.error(prog, #ID.getLineNumber(), "Looking for `" +
                    data.getType() + "', but found `" + a +"'.");
      }
   }

}

/*
 * Object method calls
```

```
 *
 * Set isTableMethod=true as necessary to ensure that expr
 * actually contains an unindexed one-dimensional array.
 */
| #(OBJECT_METHOD n:ID m:ID
   {
      data=sts.getWclData(n.getText());
      if (data.isTable()) {
         isTableMethod=true;
      }
   }
  #(ARGS (a=expr (b=expr)?)? ) {isTableMethod=false;})
{

   /* Methods common to all objects */
   if ( (data.isWebPage() || data.isParagraph() ||
         data.isLink() || data.isImage() ||
         data.isTable() ) &&
        (m.getText().equals("setProperty") ) ) {
         if (a == null) {
            Main.error(prog, n.getLineNumber(), data.getType () +
                 ".setProperty method only supports two arguments, and " +
                 "they both must be of type `" + WclData.getString() +"'.");
         } else if (b == null) {
            Main.error(prog, n.getLineNumber(), data.getType () +
                 ".setProperty method only supports two arguments, and " +
                 "they both must be of type `" + WclData.getString() +"'.");
         } else if  (!WclData.isString(a) || !WclData.isString(b) ) {
            Main.error(prog, n.getLineNumber(), data.getType () +
                 ".setProperty method only supports two arguments, " +
                 "and they both must be of type `" + WclData.getString()+
                 "', not `" + a + "' and `" + b + "'.");
         }
      }

   /* WebPage objects */
   else if (data.isWebPage()) {
      /* WebPage.add method */
      if (m.getText().equals("add")) {
         if ( a == null || b != null || (
```

```
      !WclData.isLink(a) && !WclData.isTable(a) &&
      !WclData.isImage(a) && !WclData.isParagraph(a) )
      )
      {
         Main.error(prog, n.getLineNumber(), "WebPage.add method " +
                    "only supports a single argument, and that " +
                    "argument must be one of the other objects, " +
                    "not ‘" + a + "’.");
      }
   /* WebPage.save method */
   } else if (m.getText().equals("save")) {
      if ( a == null || !WclData.isString(a) )
      {
         Main.error(prog, n.getLineNumber(), "WebPage.save method " +
                    "only supports a single argument, and that " +
                    "argument must be of type ‘" + WclData.getString() +
                    "’, not ‘" + a + "’.");
      }
   } else {
      Main.error(prog, n.getLineNumber(), "WebPage doesn’t support " +
                 "method ‘" + m.getText() + "’.");
   }

/* Paragraph objects */
} else if (data.isParagraph()) {
   /* Paragraph.add method */
   if (m.getText().equals("add")) {
      if ( a == null || b != null || !WclData.isString(a) ) {
         Main.error(prog, n.getLineNumber(), "Paragraph.add method " +
                    "only supports a single argument, and that " +
                    "argument must be string, not ‘" + a + "’.");
      }
   } else {
      Main.error(prog, n.getLineNumber(), "Paragraph doesn’t support " +
                 "method ‘" + m.getText() + "’.");
   }

/* Link objects */
} else if (data.isLink()) {
   /* Link.setURL and setText methods */
```

```
   if (m.getText().equals("setURL") || m.getText().equals("setText")) {
      if ( a == null || b != null || !WclData.isString(a) ) {
         Main.error(prog, n.getLineNumber(), "Link." + m.getText() +
                    " method only supports a single argument, and " +
                    "that argument must be ‘" + WclData.getString() +
                    "’, not ‘" + a + "’.");
      }
   /* Link.setImage method */
   } else if (m.getText().equals("setImage")){
      if ( a == null || b != null || (
          !WclData.isString(a) && !WclData.isImage(a) )
         ) {
         Main.error(prog, n.getLineNumber(), "Link.setImage method " +
                    "only supports a single argument and that " +
                    "argument must be of type either ‘" +
                    WclData.getString() + "’ or ’Image’, " +
                    "not ‘" + a + "’.");
      }
   } else {
      Main.error(prog, n.getLineNumber(), "Link doesn’t support " +
                 "method ‘" + m.getText() + "’.");
   }

/* Image objects */
} else if (data.isImage()) {
   /* Image.setImage method */
   if (m.getText().equals("setImage")) {
      if ( a == null || b != null || !WclData.isString(a) ) {
         Main.error(prog, n.getLineNumber(), "Image.setImage method " +
                    "only supports a single argument and that " +
                    "argument must be of type ‘" + WclData.getString() +
                    "’ not ‘" + a + "’.");
      }
   }

/* Table objects */
} else if (data.isTable()) {
   /* Table.add method */
   if (m.getText().equals("add")) {
      /* Make sure we received a single one-dimensional array arg */
```

```
            if (a == null || b != null || arrayData == null ||
               arrayData.getNumIndices() != 1 || arrayNumIndicesCalled != 0) {
               Main.error(prog, n.getLineNumber(), "Table.add method " +
                          "only supports a single argument and that " +
                          "argument must be an unindexed, one-dimensional " +
                          "array.");
            }
         }

      /* Method calls for all other types should be an error */
      } else {
         Main.error(prog, n.getLineNumber(), "Type `" + data.getType() +
                     "' does not support any methods.");
      }

   }
   ;


primitives
   : "void"
   | "bool"
   | "int"
   | "float"
   | "string"
   | "WebPage"
   | "Link"
   | "Paragraph"
   | "Table"
   | "Image"
   ;

objects returns [String r]
{
   r = null;
}
   : "WebPage"    { r = "WebPage"; }
   | "Paragraph"  { r = "Paragraph"; }
   | "Table"      { r = "Table"; }
   | "Image"      { r = "Image"; }
```

```
   | "Link"       { r = "Link"; }
   ;
```

A.3 WCL Code Generation

```
{
import java.io.*;
}
class WclCodeGen extends TreeParser;
options{
   importVocab = Wcl;
   ASTLabelType = "WclAST";
}

{

   String prog = "WclCodeGen";
   String a,b,c,id,type,y,d,e,f,g;
   WclAST m,n;
   int line;
   String javaClassName = "public class " + Main.getFile() + "{ \n";
   String javaCodeMain ="public static void main(String args[]) {\n";
   String javaFunctions = "";
   String globalVars = "";
   String index = null;
   boolean nextArg = false;
   boolean isFunction = false;

}


/**
 * Starts the walker for code generation
 */
codegeneration
  : #(PROG (stmt)*)
   {

   FileOutputStream out; // declare a file output object
   PrintStream p; // declare a print stream object
```

```
   if (Main.do_code) {
      System.out.println(javaClassName + globalVars + javaCodeMain +"}\n"+ javaFunctions
   } else {
      try
      {
         // Create a new file output stream
         out = new FileOutputStream(Main.getFile() + ".java");

         // Connect print stream to the output stream
         p = new PrintStream( out );
            p.println(javaClassName + globalVars + javaCodeMain +"}\n" + javaFunctions +
      }
      catch (Exception e)
      {
      System.err.println ("Error writing to file");
      }
   }




   }
   ;

/**
 * In Wcl, expressions return values.
 *
 * This function returns the type of the value.
 */
expr returns [String r]
{
   r=null;
    index = null;


}
   : #(VALUE (f=expr)?)      { r = "="+ f;}
   | #(TYPE n:primitives)
```

```
 {
 if(n.getText().equals("string"))
      r ="String";
 else if(n.getText().equals("bool")) {

      r ="boolean";
      }
 else if(n.getText().equals("float"))
      r ="double";
 else r = n.getText();
}
 | NUM_INT              { r = #NUM_INT.getText();}
 | NUM_FLOAT            { r = #NUM_FLOAT.getText(); }
 | String               { r = #String.getText(); }
 | ID        { r = #ID.getText();}
 {

    String id=#ID.getText();

 }
 | #(PLUS f=expr g=expr)
 {

    r = f + "+" + g;
 }
 | #(MINUS f=expr g=expr)
 {

    r = f + "-" + g;
 }
 | #(TIMES f=expr g=expr)
 {

    r = f + "*" + g;
 }
 | #(SLASH f=expr g=expr)
 {

    r =f + "/" + g;
 }
```

```
| #(MODULO f=expr g=expr)
{
r =f + "%" + g;
}
| #(CONDITION a=expr) { r = a;}
     | #(AND a=expr b=expr){ r = a + "&&" + b;}
     | #(OR a=expr b=expr){  r = a + "||" + b;}
     | #(LESS d=expr e=expr)
     {
     r =  d + "<" + e;
     }
     | #(MORE d=expr e=expr)
     {
      r = d + ">" + e;
     }
     | #(EQUALTO d=expr e=expr)
     {
      r =  d + "==" + e;
     }
       | #(NOTEQUAL d=expr e=expr)
      {
         r =  d + "!=" + e;
      }
     | #(LESSEQUAL d=expr e=expr)
     {
      r = d + "<=" + e;
     }
     | #(MOREEQUAL d=expr e=expr)
     {
     r = d + ">=" + e ;
}

| #(INDEX b=expr (y=expr)?)
{
     if(y==null){
             r = "[" + b + "]";
             index = "[] ";
     }
     else{
             r = "[" + b + "]" + "[" + y + "]" ;
```

```
                index = "[][] ";
        }


   }
  | #(CALL_FUNCTION ID {


        r = #ID.getText() + "(";

  }

   #(ARGS (d=expr

   {
   if(nextArg == true)
   {

        r +=", ";
   }
   r +=  d;
   nextArg = true;
   })*)
  {
  r += ")";

nextArg = false;
})

  ;

stmt
{
   a=null; b=null; c=null; type=null; y=null; d=null; e=null;f=null;
}



   /*
    * Variable declaration with optional assignment
```

```
 */
: #(VARIABLE a=expr ID (b=expr)? )
{
id = #ID.getText();
if(isFunction == false){
        if (index != null){
             javaCodeMain += a + index + id +  "= new " + a + b  + ";\n";
        }
        else if(b != null){

           javaCodeMain +=  a + " " + id +  b  + ";\n";
        }
        else if(a.equals("String")){
           javaCodeMain +=  a + " " + id +  "=\"\";\n";
        }
        else if(a.equals("int")){
           javaCodeMain += a +" " + id + "=0;\n";
        }
        else if(a.equals("WebPage")){
           javaCodeMain += a +" " + id + "= new WebPage();\n";
        }
        else if(a.equals("Link")){
           javaCodeMain += a +" " + id + "= new Link();\n";
        }
        else if(a.equals("Image")){
              javaCodeMain += a +" " + id + "= new Image();\n";
        }
        else if(a.equals("Paragraph")){
              javaCodeMain += a +" " + id + "= new Paragraph();\n";
        }
        else if(a.equals("Table")){
                     javaCodeMain += a +" " + id + "= new Table();\n";
        }
        else if(a.equals("double")){
                 javaCodeMain += "double" +" " + id + "= 0.0;\n";
        }
        else if(a.equals("boolean")){
                    javaCodeMain += "boolean "  + id + "= false;\n";
        }
     }
```

```
   else {
          if (index != null){
               javaFunctions += a + index + id +  "= new " + a + b  + ";\n";
          }
          else if(b != null){

             javaFunctions +=  a + " " + id +  b  + ";\n";
          }
          else if(a.equals("String")){
             javaFunctions +=  a + " " + id +  "=\"\";\n";
          }
          else if(a.equals("int")){
             javaFunctions += a +" " + id + "=0;\n";
          }
          else if(a.equals("WebPage")){
             javaFunctions += a +" " + id + "= new WebPage();\n";
          }
          else if(a.equals("Link")){
             javaFunctions += a +" " + id + "= new Link();\n";
          }
          else if(a.equals("Image")){
                javaFunctions += a +" " + id + "= new Image();\n";
          }
          else if(a.equals("Paragraph")){
                javaFunctions += a +" " + id + "= new Paragraph();\n";
          }
          else if(a.equals("Table")){
                javaFunctions += a +" " + id + "= new Table();\n";
          }
          else if(a.equals("double")){
                javaFunctions += "double" +" " + id + "= 0.0;\n";
          }
          else if(a.equals("boolean")){
                javaFunctions += "boolean "  + id + "= false;\n";
          }

   }
  }
/*
   * Variable assignment
```

```
   */
  | #(STATEMENT ID (y=expr)? EQUAL a=expr)
  {

     id = #ID.getText();

     if(isFunction == false){
             if(y==null){
               javaCodeMain += id +"=" + a + ";\n";
             }
             else{
               javaCodeMain += id +y+"=" + a + ";\n";
             }
    }
    else{

             if(y==null){
               javaFunctions += id +"=" + a + ";\n";
             }
             else{
               javaFunctions += id +y+"=" + a + ";\n";
             }

    }
  }

  |#(RETURN_VAL y=expr)
  {
       javaFunctions += "return " + y + ";\n";

  }

/*
   * Global variable declaration with optional assignment
   */
  | #(GLOBAL #(VARIABLE a=expr ID (b=expr)?  ) )
  {
       id = #ID.getText();
       if (index != null){
               globalVars += "static " + a + index + id +  "= new " + a + b  + ";\n";
```

```
    }
    else if(b != null){

       globalVars +=  "static "+ a + " " + id +  b  + ";\n";
    }
    else if(a.equals("String")){
       globalVars +=  "static "+ a + " " + id +  "=\"\";\n";
    }
    else if(a.equals("int")){
       globalVars += "static "+ a +" " + id + "=0;\n";
    }
    else if(a.equals("WebPage")){
       globalVars += "static "+ a +" " + id + "= new WebPage();\n";
    }
    else if(a.equals("Link")){
       globalVars += "static "+ a +" " + id + "= new Link();\n";
    }
    else if(a.equals("Image")){
          globalVars += "static "+ a +" " + id + "= new Image();\n";
    }
    else if(a.equals("Paragraph")){
          globalVars += "static "+ a +" " + id + "= new Paragraph();\n";
    }
    else if(a.equals("Table")){
                  globalVars += "static "+ a +" " + id + "= new Table();\n";
    }
    else if(a.equals("double")){
          globalVars += "static "+ "double" +" " + id + "= 0.0;\n";
    }
    else if(a.equals("bool")){
          globalVars += "static "+ "boolean"  + id + "= false;\n";
    }


 }
 /*
 * Function declaration
 */
| #(DECLARE #(FUNCTION "function" d=expr ID ARGS) )
{ /*do nothing*/}
```

```
  | #(FUNCTION "function" {javaFunctions += "public static ";} d=expr {javaFunction

   {
     id = #ID.getText();

     {
     javaFunctions += " " + id + "(";
     nextArg = false;
     isFunction=true;

     }


  }

 #(ARGS  (d=expr {if(nextArg == true){ javaFunctions += ", ";} javaFunctions += d;}
   {
   javaFunctions += ")\n";
   nextArg = false;
   })



  /*
   * Ok, now continue matching rest of function body
   */
    #(FUNCTION_BODY {javaFunctions += "{\n";} (stmt)* (#(RETURN_VAL d=expr) {javaFu

    {
    javaFunctions += "}\n";
    isFunction = false;

    }
)

/*
   * Function call
   */
```

```
| #(CALL_FUNCTION ID {

if(isFunction ==true)
 {
         javaFunctions += #ID.getText() + "(";
}
else{
         javaCodeMain += #ID.getText() + "(";
}

}

 #(ARGS (e=expr

 {
 if(nextArg == true)
 {
         if(isFunction ==true)
         {
                 javaFunctions +=", ";
         }
         else
         {
                 javaCodeMain += ", ";
         }
 }
 if(isFunction == true){
         javaFunctions +=  e;

 }
 else{
         javaCodeMain +=  e;

 }
 nextArg = true;
 })*)
{
 if(isFunction == true){
         javaFunctions += ");\n";
 }
```

```
else{
        javaCodeMain += ");\n";
}
nextArg = false;
})



 /*
        * object method call
        */
       | #(OBJECT_METHOD n:ID m:ID {

       if(isFunction ==true)
        {
                javaFunctions += n.getText() +"." + m.getText() + "(";
       }
       else{
                javaCodeMain += n.getText() +"." + m.getText() +  "(";
       }

       }

        #(ARGS (a=expr

        {
        if(nextArg == true)
        {
                if(isFunction ==true)
                {
                        javaFunctions +=", ";
                }
                else
                {
                        javaCodeMain += ", ";
                }
        }
        if(isFunction == true){
                javaFunctions +=  a;
```

```
        }
        else{
                javaCodeMain +=  a;

        }
        nextArg = true;
        })*)
       {
        if(isFunction == true){
                javaFunctions += ");\n";
        }
        else{
                javaCodeMain += ");\n";
        }
        nextArg = false;
        })




/*
  * While loop, need new scope
  */
 | #("while" c=expr {javaCodeMain += "while " + "(" + c + ")" + "{\n" ;} (stmt)*

 /*
  * Conditional if, then, else, need new scope
  */
 | #("if" c=expr {

        if(isFunction ==true)
        {
                javaFunctions += "if " + "(" + c + ")" + "{\n";

        }
        else{
                javaCodeMain += "if " + "(" + c + ")" + "{\n";
        }
 }
```

```
        #(THEN (stmt)* {
        if(isFunction ==true)
              {
              javaFunctions +=  "}\n";
              }
              else{
              javaCodeMain +=  "}\n";
              }
        })
        (#("else" {
              if(isFunction ==true){
               javaFunctions += "else {\n";
               }
               else{
                      javaCodeMain += "else {\n";
              }
              }(stmt)*
              {
             if(isFunction ==true){
              javaFunctions += "}\n";
             }
             else{
               javaCodeMain +="}\n";
             }



              }))?
         )
      ;



primitives
   : "void"
   | "bool"
   | "int"
   | "float"
   | "string"
```

```
   | "WebPage"
   | "Link"
   | "Paragraph"
   | "Table"
   | "Image"
   ;

objects returns [String r]
{
   r = null;
}
   : "WebPage"    { r = "WebPage"; }
   | "Paragraph"  { r = "Paragraph"; }
   | "Table"      { r = "Table"; }
   | "Image"      { r = "Image"; }
   | "Link"       { r = "Link"; }
   ;

methods returns [String r]
{
   r = null;
}
   : "add"    { r = "add"; }
   | "set"    { r = "set"; }
   ;
```

B WCL Java Code

Below is the collection of java files used by the WCL compiler (in alphabetical order):

Image.java

Link.java

Main.java

```
import java.io.*;
import antlr.DumpASTVisitor;
```

```
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
public class Main {

   public static String file;
   public static String type;
   public static boolean do_gram = false;
   public static boolean do_sem  = false;
   public static boolean do_code = false;
   public static boolean do_ast  = false;

   public static void main(String args[]) {

      /* Make sure we've got a file to process */
      if (args.length == 1) {
         file = args[0];
      } else if (args.length == 2) {
         /* Our secret switch for testing */
         file = args[0];
         type = args[1];
         if (type.equals("gram")) {
            do_gram = true;
         } else if (type.equals("ast")) {
            do_ast = true;
         } else if (type.equals("sem")) {
            do_sem = true;
         } else if (type.equals("code")) {
            do_code = true;
         }
         else {
            System.err.println ("Usage: java Main filename");
            System.err.println ("Specify exactly one file on command line");
            System.exit(-1);
         }
      } else {
         System.err.println ("Usage: java Main filename");
         System.err.println ("Specify exactly one file on command line");
         System.exit(-1);
      }
```

```
/* Start compiling */
try {
   /* Read in the file */
   FileInputStream input = new FileInputStream(file);

   /* Do lexing and parsing */
   boolean validSyntax = true;
   WclLexer lexer = new WclLexer(input);
   WclParser parser = new WclParser(lexer);

   try {
   parser.program();
   input.close();
   } catch (Exception e) {
      System.out.println("Syntax error at: " + e.getMessage());
      validSyntax = false;
      System.exit(0);
   }
   WclAST parseTree = (WclAST)parser.getAST();

   /* Print out AST and exit if we're testing */
   if (do_gram || !validSyntax) {
      System.out.println(parseTree.toStringList());
      System.exit(0);
   } else if (do_ast) {
      ASTFrame parseFrame = new ASTFrame("Parser's Tree", parseTree);
      parseFrame.setVisible(true);
   }

   /* Do semantic analysis */
   if (!do_gram && !do_ast) {
      WclSemanticCheck walker = new WclSemanticCheck();
      walker.checkSemantics(parseTree);
   }

   /*Do code generation*/
   //System.out.println("\n****Begin code generation");
   if (!do_gram && !do_sem && !do_ast) {
```

```
            WclCodeGen codegen = new WclCodeGen();
            codegen.codegeneration(parseTree);
         }


      } catch(Exception e) { System.err.println("Exception in Main: "+e); }
   }

   public static void error(String prog, int lineNum, String strMessage) {
      try {
         String line="";
         BufferedReader input = new BufferedReader(new FileReader(file));
         for (int i=0;i<lineNum;i++) {
            line = input.readLine();
         } if (lineNum > 0) {
         System.out.println(prog + ": Error on line " + lineNum + ".");
         System.out.println(prog + ": " + line);
         }
         System.out.println(prog + ": " + strMessage);
         System.exit(-1);
      } catch(Exception e) { System.err.println("Exception in error method: "+e); }
   }
   public static String getFile(){
      String changeSlash = file.replaceAll("/", "\\\\");
      String woExt = changeSlash.replaceAll("\\.", "\\\\");
      String[] newFile = woExt.split("\\\\");
      return newFile[newFile.length -2]; //name of file without directories or extension
   }
}
```

Paragraph.java

```
/*
 * Class for html paragraphs
 * @author Sarah Friedman sf2179
 */


public class Paragraph {
        private String text = "";
```

```
        private String font = "<FONT ";
        public void add(String txt){
                String htmlText = txt.replaceAll("\n","<BR>");
                text += htmlText;
        }

        public String toString(){
                return font + "> <P>" + text + "</P></FONT>\n";
        }
        /*set properties  -- example: size, color*/
        public void setProperty(String property, String value){
                font += property + "='" + value + "' ";
        }
}
```

Table.java

```
/*
 * Class for html images
 * @author Sarah Friedman sf2179
 */

public class Table {
        private String htmlTable ="";//start table tags
        private String properties = "";

        public void add(String row[]){  //add just one row
                htmlTable += "<tr>"; //new row
                for(int i=0;i<row.length;i++){
                        htmlTable += "<td>" +row[i] + "</td>"; //add each cell
                }
                htmlTable += "</tr>";
        }

        public void add(int row[]){
                htmlTable += "<tr>"; //new row
                                for(int i=0;i<row.length;i++){
                                        htmlTable += "<td>" +row[i] + "</td>"; //add eac
                                }
                htmlTable += "</tr>";
```

```
        }

        public void add(float row[]){
                htmlTable += "<tr>"; //new row
                                for(int i=0;i<row.length;i++){
                                        htmlTable += "<td>" +row[i] + "</td>"; //add eac
                                }
                htmlTable += "</tr>";
        }


        public String toString(){
                return "<table " + properties + ">" + htmlTable + "</table>\n";
        }

        /*set table properties  -- example: border, height*/
        public void setProperty(String property, String value){
                properties += property + "='" + value + "' ";
        }
}
```

WclAST.java

```
/**
 * The AST node.  Includes line number information.
 *
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */

import antlr.*;
import antlr.collections.AST;

public class WclAST extends CommonAST
{
   private int lineNumber;

   public void initialize(AST t) {
      super.initialize(t);
      if (t.getClass().getName().compareTo("WclAST") == 0)
      {
         WclAST t2 = (WclAST)t;
```

```
         lineNumber = t2.lineNumber;
      }
   }

   public void initialize(Token tok) {
      super.initialize(tok);
      lineNumber = tok.getLine();
   }

   public int getLineNumber() {
      return lineNumber;
   }

}
```

WclData.java

```
/**
 * Data type to store in symbol table.
 * Includes several static convenience functions..
 *
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */

import java.util.Vector;

public class WclData
{
   private String type;
   private String category;
   private Vector args=null;
   private int indices=0;


   public WclData(String type, String category)
   {
      assert type != null && type != "";
      assert category != null && category != "";
      this.type = type;
      this.category = category;
   }
```

```
public WclData(String type, String category, Vector args)
{
   assert type != null && type != "";
   assert category != null && category != "";
   assert args != null;
   this.type = type;
   this.category = category;
   this.args = args;
}

public WclData(String type, String category, int indices)
{
   assert type != null && type != "";
   assert category != null && category != "";
   assert indices > 0;
   this.type = type;
   this.category = category;
   this.indices = indices;
}

public String getType()
{
   assert type != null && type != "";
   return type;
}

public Vector getArgs()
{
   assert args != null;
   return args;
}

public int getNumIndices()
{
   return indices;
}

public boolean isVariable()
{
```

```
   if (category.equals("variable"))  { return true; }
   return false;
}

public boolean isGlobal()
{
   if (category.equals("global"))    { return true; }
   return false;
}

public boolean isArray()
{
   if (category.equals("array"))     { return true; }
   return false;
}

public boolean isFunction()
{
   if (category.equals("function"))  { return true; }
   return false;
}

public boolean isWebPage()
{
   if (type.equals("WebPage"))   { return true; }
   return false;
}

public boolean isParagraph()
{
   if (type.equals("Paragraph")) { return true; }
   return false;
}

public boolean isLink()
{
   if (type.equals("Link"))      { return true; }
   return false;
}
```

```
public boolean isImage()
{
   if (type.equals("Image"))     { return true; }
   return false;
}

public boolean isTable()
{
   if (type.equals("Table"))     { return true; }
   return false;
}



public static boolean isArray(String str)
{
   if (str.equals("array"))       { return true; }
   return false;
}

public static boolean isInt(String str)
{
   if (str.equals("int"))       { return true; }
   return false;
}

public static boolean isString(String str)
{
   if (str.equals("string"))    { return true; }
   return false;
}

public static boolean isFloat(String str)
{
   if (str.equals("float"))     { return true; }
   return false;
}

public static boolean isBool(String str)
{
```

```
   if (str.equals("bool"))     { return true; }
   return false;
}

public static boolean isWebPage(String str)
{
   if (str.equals("WebPage"))   { return true; }
   return false;
}

public static boolean isParagraph(String str)
{
   if (str.equals("Paragraph")) { return true; }
   return false;
}

public static boolean isLink(String str)
{
   if (str.equals("Link"))      { return true; }
   return false;
}

public static boolean isImage(String str)
{
   if (str.equals("Image"))     { return true; }
   return false;
}

public static boolean isTable(String str)
{
   if (str.equals("Table"))     { return true; }
   return false;
}

public static String getInt()
{
   return "int";
}

public static String getString()
```

```
   {
      return "string";
   }

   public static String getFloat()
   {
      return "float";
   }

   public static String getBool()
   {
      return "bool";
   }

   public static String getVariable()
   {
      return "variable";
   }

   public static String getFunction()
   {
      return "function";
   }

   public static String getArray()
   {
      return "array";
   }

   public static String getNull()
   {
      return "NULL";
   }

}
```

WclFunctionArg.java

```
/**
 * Data type to store function arguments/parameters..
 *
```

```
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */

public class WclFunctionArg {

   static final int ARRAY = 1;
   static final int VARIABLE = 2;

   private int category;
   private String type;
   private int arrayDimensions;

   public WclFunctionArg(String type, int category) {
      assert type != null && type != "";
      assert category > 0 && category != ARRAY;
      this.category = category;
      this.type = type;
   }

   public WclFunctionArg(String type, int category, int dimensions) {
      assert type != null && type != "";
      assert category > 0 && category == ARRAY;
      assert dimensions > 0;
      this.category = category;
      this.type = type;
      this.arrayDimensions = dimensions;
   }

   public String getType() {
      assert type != null && type != "";
      return type;
   }

   public int getCategory() {
      assert category > 0;
      return category;
   }

   public int getDimensions() {
      assert arrayDimensions > 0;
```

```
      return arrayDimensions;
   }

}
```

WclReservedWords.java

```
import java.util.HashSet;
import java.util.Arrays;

public class WclReservedWords
{
   /* Taken from http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.htm
   private static String[] reservedArray = {
      "abstract", "double", "int", "strictfp", "boolean", "else", "interface",
      "super", "break", "extends", "long", "switch", "byte", "final", "native",
      "synchronized", "case", "finally", "new", "this", "catch", "float",
      "package", "throw", "char", "for", "private", "throws", "class", "goto",
      "protected", "transient", "const", "if", "public", "try", "continue",
      "implements", "return", "void", "default", "import", "short", "volatile",
      "do", "instanceof", "static", "while"
   };

   /* Convert array to HashSet for easy lookup */
   private static HashSet reservedSet = new HashSet(Arrays.asList(reservedArray));

   public static boolean isReserved(String word) {
      assert (word != null & !word.equals(""));
      return reservedSet.contains(word);
   }

}
```

WclSymbolTable.java

```
/**
 * Symbol table class with static scoping.
 *
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */
```

```
import java.util.*;

class WclSymbolTable extends HashMap {
   private WclSymbolTable parent;

   public WclSymbolTable(WclSymbolTable parent) {
      this.parent = parent;
   }

   public WclSymbolTable getParent() {
      return parent;
   }

   public boolean hasId(String id) {
      assert id != null && id != "";
      return containsKey(id);
   }

   public void addId(String id, WclData data) {
      assert id != null && data != null;
      this.put(id,data);
   }

   public WclData getWclData(String id) {
      assert id != null && id != "";
      WclSymbolTable st = this;
      Object x = st.get(id);
      while (x == null && st != null) {
        try {
            st = st.getParent();
            x = st.get(id);
         } catch (Exception e) {}
      }
      return (WclData)x;
   }

   public WclData getWclDataInFunction(String id) {
      assert id != null && id != "";
      WclSymbolTable st = this;
      WclSymbolTable test = null; //probes symbol table to throw exception.
```

```
      Object x = st.get(id);
      while (x == null && st != null) {
        try {
            st = st.getParent();
            /* Probe to see if we should exit */
            try { test = st.getParent().getParent(); }
            catch (Exception e) { return null; }
            x = st.get(id);
         } catch (Exception e) {}
      }
      return (WclData)x;
   }

}
```

WclSymbolTableStack.java

```
/**
 * The stack to support static scoping.
 *
 * @author Matthew Mintz-Habib - mm2571@cs.columbia.edu
 */

import java.util.*;

class WclSymbolTableStack {
   public Stack theStack = new Stack();
   private HashMap globals = new HashMap();
   public int count = 0;

   public WclSymbolTableStack() {
      theStack.push(new WclSymbolTable(null));
      count++;
   }

   public void enterNewScope() {
      WclSymbolTable parent = (WclSymbolTable)theStack.peek();
      assert parent != null;
      theStack.push(new WclSymbolTable(parent));
      count++;
   }
```

```
public void exitNewScope() {
   assert count > 1;
   theStack.pop();
   count--;
}

public boolean hasLocal(String id) {
   assert id != null && id != "";
   return ((WclSymbolTable)theStack.peek()).hasId(id);
}

public boolean hasGlobal(String id) {
   assert id != null && id != "";
   assert globals != null;
   return globals.containsKey(id);
}

public void addId(String id, WclData data) {
   assert id != null && data != null;
   WclSymbolTable st = (WclSymbolTable)theStack.peek();
   assert st != null;
   st.addId(id,data);
}

public void addGlobal(String id, WclData data) {
   assert id != null && id != "";
   assert data != null;
   assert globals != null;
   globals.put(id,data);
}

/**
 * Returns null if id is not found.
 */
public WclData getWclData(String id) {
   WclSymbolTable st = (WclSymbolTable)theStack.peek();
   assert st != null;
   WclData result = st.getWclData(id);
   if (result == null) {
```

```
            result = getGlobal(id);
         }
         return result;
      }

      /**
       * Returns null if id is not found.
       * This function is used to check ID scoping inside function definitions.
       * Functions are in the second level scope.  If an ID is not found inside
       * the function scope, don't look in first level, look in global scope
       * instead.  This solves the problem of function calls referencing
       * variables that haven't been declared yet.
       */
      public WclData getWclDataInFunction(String id) {
         WclSymbolTable st = (WclSymbolTable)theStack.peek();
         assert st != null;
         WclData result = st.getWclDataInFunction(id);
         if (result == null) {
            result = getGlobal(id);
         }
         return result;
      }

      /**
       * May return null
       */
      public WclData getGlobal(String id) {
         assert globals != null;
         return (WclData)globals.get(id);
      }

      public WclSymbolTable getCurrentSymbolTable() {
         return (WclSymbolTable)theStack.peek();
      }


}
```

WebPage.java

```
/*
```

```
 * Class for html WebPages
 * @author Sarah Friedman sf2179
 */

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

public class WebPage {
        private String webpage = "<HTML>";
        private Vector webPageObjects = new Vector();
        public WebPage(String title){ //constructor with title
                webpage += "<HEAD><TITLE>" + title + "\n</TITLE></HEAD>\n<BODY>\n";
        }
        public WebPage(){ //constructor with no arguments
                webpage += "\n<HEAD></HEAD>\n<BODY>\n";
        }

        public void add(Table table){
                webPageObjects.add(table);
                //webpage += table.toString();
        }
        public void add(Image img){
                webPageObjects.add(img);
                //webpage += img.toString();
        }
        public void add(Link link){
                webPageObjects.add(link);
                //webpage += link.toString();
        }
        public void add(Paragraph para){
                webPageObjects.add(para);
                //webpage += para.toString();
        }
        public void save(String pageName){
                Object[] obj = webPageObjects.toArray();
                for (int i=0;i<obj.length;i++){
                        webpage += obj[i].toString();
                }
                FileOutputStream out; // declare a file output object
        PrintStream p; // declare a print stream object
```

```
        try
        {
                // Create a new file output stream
                out = new FileOutputStream(pageName);

                // Connect print stream to the output stream
                p = new PrintStream( out );

                p.println (webpage);  //print the webpage

                p.close();
        }
        catch (Exception e)
        {
                System.err.println ("Error writing to file");
        }

        }
}
```