HGL: Hypertext Generation Language

Yan Koyfman (ysk2106@columbia.edu)
Shruti Gandhi (slg2109@columbia.edu)
Timothy Kaczynski (tdk2102@columbia.edu)
Edward Mezarina (eem65@columbia.edu)

December 20, 2005

COMS W4115: Programming Languages and Translators
Department of Computer Science
Columbia University

Table of Contents

1.0 INTRODUCTION 5

1.1 Goals 5

1.2 Main Language Features 5
- 1.2.1 Syntax 5
- 1.2.2 Data Types 6
- 1.2.3 Flow Control 6
- 1.2.4 Looping 6
- 1.2.5 Operators 6
- 1.2.6 Scoping 6
- 1.2.7 Comments 6
- 1.2.8 Example 7

2.0 TUTORIAL 8

2.1 Getting Started 8
- 2.1.1 Install HGL 8
- 2.1.2 Creating an HGL program 9
- 2.1.3 Generate Your Web Page 10

2.2 Advanced Examples 10
- 2.2.1 Tables, Links, and Images 10
- 2.2.2 Functions/Table/Loops/Conditionals 13
- 2.2.3 Arithmetic Operations in HGL 14
- 2.2.4 Real-Life Example 16

3.0 LANGUAGE REFERENCE MANUAL 18

3.1 Grammar Notation 18
- 3.1.1 Minimum File 18

3.2 Lexical Conventions 18
- 3.2.1 Comments 18
- 3.2.2 Whitespace 18
- 3.2.3 Identifiers 18
- 3.2.4 Keywords 18
- 3.2.5 Strings 19
- 3.2.6 Operators 19
- 3.2.7 Other Tokens 19
- 3.2.8 Types 20
- 3.2.9 Expressions 21
- 3.2.10 Statements 23

3.3 HGL Library 25
- 3.3.1 Built-in HTML Objects and Properties 25

4.0 PROJECT PLAN 29

4.1 Process used **29**
4.1.1 Planning and specification 29
4.1.2 Development and Testing 29

4.2 Team Responsibilities **29**

4.3 Programming Style **29**

4.4 Project Timeline **30**

4.5 Software Project Environment **30**
4.5.1 Software 30
4.5.2 Tools 30

4.6 Project Log **31**

5.0 ARCHITECTURE DESIGN **32**

5.1 Front End **33**

5.2 Back End **33**

5.3 Logger and Error Handler **34**

6.0 TESTING PLAN **36**

6.1 Unit Test **36**

6.2 Function Test **36**

6.3 Regression Test **39**
6.3.1 Test bucket 39
6.3.2 Automation 39
6.3.3 Examples 40

7.0 LESSONS LEARNED **43**

7.1 Shruti Gandhi **43**

7.2 Yan Koyfman **43**

7.3 Tim Kaczynski **43**

7.4 Edward Mezarina **44**

8.0 BIBLIOGRAPHY **45**

9.0 APPENDIX A: CSS PROPERTIES SUPPORTED BY HGL [2],[3] **46**
9.1.1 Font 46
9.1.2 Font Style 46
9.1.3 Font Size 47
9.1.4 Font Family 48

9.1.5 Border 49
9.1.6 Left Border 49
9.1.7 Bottom Border 50
9.1.8 Right Border 50
9.1.9 Top Border 51
9.1.10 Width 51
9.1.11 Height 52
9.1.12 Float 52
9.1.13 Top Margin 53
9.1.14 Right Margin 53
9.1.15 Bottom Margin 54
9.1.16 Left Margin 54
9.1.17 Margin 55
9.1.18 Top Padding 55
9.1.19 Right Padding 56
9.1.20 Bottom Padding 56
9.1.21 Left Padding 56
9.1.22 Padding 57
9.1.23 Top 57
9.1.24 Right 58
9.1.25 Bottom 58
9.1.26 Left 59
9.1.27 Background 59
9.1.28 Line Height 60
9.1.29 Color (hgl: fgcolor) 60
9.1.30 Background Color (HGL: bgcolor) 61
9.1.31 Background Image 61
9.1.32 Vertical Alignment 62
9.1.33 Position 63
9.1.34 Z-index 63
9.1.35 Text Alignment 64

10.0 APPENDIX B: CODE LISTINGS 65

10.1 Examples 65
10.1.1 pbc.hgl 65

10.2 HGL Translator Code Listing 73
10.2.1 Parser 73
10.2.2 HGL Exceptions 91
10.2.3 HGL Logging and Related Classes 95
10.2.4 HGL HTML Objects, CSS Generation and Related Code 96

1.0 Introduction

There was a time when a person with an idea or an opinion had to convince a publisher that the content they had created was worth the time and money required to distribute it. The Internet has made it possible for anybody with a personal computer to publish just about whatever they want to an incredibly large and diverse community of people, at little to no cost. Unfortunately, the languages that describe to a computer how to display that content have little to nothing in common with the languages used for general purpose computing. HGL allows a programmer to display his or her content using familiar syntax and semantics.

HyperText Markup Language, or HTML, is, as its name suggests, a markup language where content is static and buried in a sea of tags which make it difficult to edit. Languages such as PHP provide more flexibility and a structure which more closely resembles C or Java, but still require the programmer to have an understanding of how HTML works. HGL is a language that borrows some of the syntax of PHP and C but which abstracts the markup language of HTML inside built-in data types, so the programmer can concentrate on their page's content and presentation using a simple set of primitive objects without having to remember dozens of confusing HTML tags or CSS properties.

HGL is geared towards the experienced C or Java programmer who desires a simple and familiar way to create web pages which contain static and a limited amount of dynamic content. HGL, when compiled, produces a mix of HTML and CSS (Cascading Style Sheets, defining the page's visual characteristics) which can then be directly published to the Internet.

1.1 Goals

The main goal of HGL is to remove the dependency on HTML from Web-oriented languages such as PHP. When used for web development, a drawback of PHP is the requirements to embed HTML tags as hard-coded strings directly inside the PHP code. While PHP provides the ability to easily turn static content into dynamic content, it requires an extensive knowledge of HTML and CSS on the part of the developer to handle presentation logic. HTML is a language which has little to nothing in common with recent high level languages for general purpose computing, such as C and Java. This requires someone who is already skilled in general purpose computing and programming to learn yet another language. HGL contains built-in types to abstract the presentation logic of HTML from the developer, allowing them to describe their content using familiar syntax and semantics.

PHP is an incredibly powerful language in and of itself. There are also an extensive list of built-in and third-party libraries available to add features such as database and EIS access. HGL does not attempt to duplicate the breadth of PHP. Instead, HGL borrows some syntactical features of the PHP language, and adds features that provide the integrated Web presentation logic which PHP lacks. Unlike PHP, HGL is not meant to be embedded in web pages. It is a stand-alone interpreted language that generates a complete Web page as its output.

1.2 Main Language Features

This section is a brief introduction to the language. More details are provided in section 2.0, the Tutorial, and in section 3.0, the Language Reference Manual.

1.2.1 Syntax

The syntax of HGL resembles that of PHP, with modifications. Variable names are preceded by a dollar sign ($), but must be declared before use. Some features present in PHP are not, permitted in HGL; for example, variable interpolation in double quoted strings, hashes and several looping structures (the syntactic sugar), are not available.

1.2.2 Data Types

Support for standard types, such as integer, floating point, boolean and strings, are available. In addition, built-in objects to represent HTML constructs such as a page, paragraph, table, ordered and unordered list are supported, along with several settable attributes per object. All variables must be declared before use, along with their type. Arrays of one dimension of all objects are possible. Dynamic arrays, however, are not available; array sizes must also be declared before use, although expressions incorporating variables and even function calls can be used as the size parameter when array objects are declared. Arrays can be indexed only with integers, not strings.

1.2.3 Flow Control

Standard mechanisms for controlling if/else statements are available. The body of loops is delimited by curly braces, { and }.

1.2.4 Looping

HGL allows 'while' style loops.

1.2.5 Operators

The following arithmetic operators are permitted: + - / *. In addition, a special string concatenation operator, | (pipe), is available. The period (.) operator is used to access attributes and methods on HTML objects. Standard Boolean operators for use in if/else statements are present as well: < >, <=, >=, ==, !=, ||, &&.

1.2.6 Scoping

Symbols are scoped at either file (global) level or the block level, depending on where they are declared. Blocks are delimited by curly braces, { and }. Each block introduces a new scope, not visible to objects in the parent scope.

1.2.7 Comments

Embedded comments are prefixed with two forward slash characters (//), which comments the remainder of the line.

1.2.8 Example

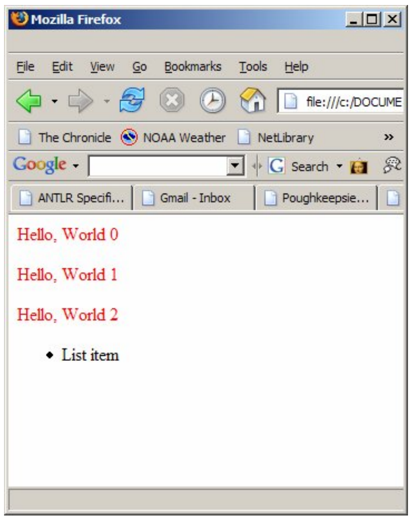
The HGL code in Figure 1 will generate a simple web page with three red paragraphs and an unordered list, as show in Figure 2 and Figure 3. The HTML produced is laid out, with the help of nesting-level based indentation, such that it is easy to use as a template for adding more content.

```
// Define simple variables
int i;
string red;
string str;

// Declare HTML objects; several are built
in to the language
Page pg1;
Paragraph pp[3];
UList ul;

// Set values using assignment operator
$i = 0;
$red = "red";
$str = "World";

// Loop through paragraph array to set its
attributes
$i=0;
while ($i<3) {
  $pp[$i].fgcolor=$red;
  $pp[$i].add("Hello, " | $str | " " |$i);

  // Add each paragraph to the page object
  $pg1.add($pp[$i]);
  $i = $i+1;
}

// Add a list item and append to page
$ul.addItem("List item");
$pg1.add($ul);

// Print out the HTML
$pg1.print();
```

Figure 1: HGL "Hello, World" Example

```
<html><head>
<style type="text/css" title="hglCSS"
media="all">
p.Paragraph0
{
  color: 0xff0000;
}
p.Paragraph1
{
  color: red;
}
p.Paragraph2
{
  color: red;
}
</style>
</head>
<body>
  <p class="Paragraph0">
    Hello, World 0
  </p>

  <p class="Paragraph1">
    Hello, World 1
  </p>

  <p class="Paragraph2">
    Hello, World 2
  </p>

  <ul>
    <li>
      List item
    </li>

  </ul>
</body>
</html>
```

Figure 2: Corresponding HTML/CSS


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///c:/DOCUME
The Chronicle NOAA Weather NetLibrary
Google Search
ANTLR Specifi... Gmail - Inbox Poughkeepsie...

Hello, World 0

Hello, World 1

Hello, World 2

- List item

Figure 3: Sample web page.

2.0 Tutorial

2.1 Getting Started

There are three basic steps for constructing a webpage with HGL:

2.1.1 Install HGL

2.1.1.1 Building HGL

1. uncompress the source archive into an empty directory
2. ensure that you have a JDK version 1.4 or above
3. verify that ANTLR is available and can be run with the command “antlr”
4. make sure the current directory and antlr.jar are in the CLASSPATH
5. run `make`: this will run antlr to create the Java code for the translator’s scanner and parser from HGLScanner.g, and will then compile the ANTLR-generated files along with the rest of the HGL Java code into .class files and an hgl.jar file.

2.1.1.2 Running HGL

Make sure hgl.class and antlr.jar are in your class path and type the following commands at the command prompt to create HTML from HGL input files:

```
$ java hgl <OPTIONS> <HGL SOURCE>
```

<OPTIONS>:
-p : Displays CSS properties supported by HGL.
See Reference manual (section 3.3: HGL Library) for more details.
-o <FILE NAME> : Outputs the hgl produced HTML/CSS code to the specified file.
Note that if this option is not specified, the HGL produced code will be output to the console.
<HGL SOURCE> : HGL only supports one input file at a time. The filename may be left out if –p is specified.

2.1.2 Creating an HGL program

Code the contents of what you desire to appear in your web page following the recommendations in the reference manual. Here is a simple example of how to create an HGL file:

myHGLPage.hgl

```
 1   // Declarations
 2   Paragraph pr;
 3   Page p;

 4   // Add content to the paragraph.
 5   $pr.font-size="24pt";
 6   $pr.add("My First Paragraph!");

 7   // Add the paragraph to the page.
 8   $p.add($pr);

 9   // Print the web page contents
10   $p.print();
```

This example shows two HTML objects being used, Paragraph and Page. These are special HTML objects that are converted to their HTML tags by the translator. In addition, line 5 shows the use of the dot operator set a property, in this case the font-size of text added to the paragraph. Lines 6 and 8 shows the special add() function that is available on all HTML objects. It takes one argument, which can be any valid expression that, when evaluated, yields a number, string, or another HTML object. The one limitation of the add() operation is that Page objects cannot be added to another HTML object. Otherwise, any combination of HTML objects can be added to one another, nested to any level.

HGL tries to make the output readable to a human and suitable for use as a template for further manual editing by indenting all objects added based on their nesting level to increase legibility.

The HTML produced from this example is shown in Figure 4. One of the advantages of HGL is that it hides most of the intricacies of the HTML-CSS system, though in this example you can see that HGL created a CSS class for the Paragraph, where its font-size is defined. In later examples, we’ll show how objects can inherit CSS properties from other objects, and also be linked to other objects to completely assume their formatting.

```
<html><head>
<style type="text/css" title="hglCSS" media="all">
p.Paragraph0
{
  font-size: 24pt;
}
</style>
</head>
<body>
  <p class="Paragraph0">
    My First Paragraph!
  </p>

</body>

</html>
```

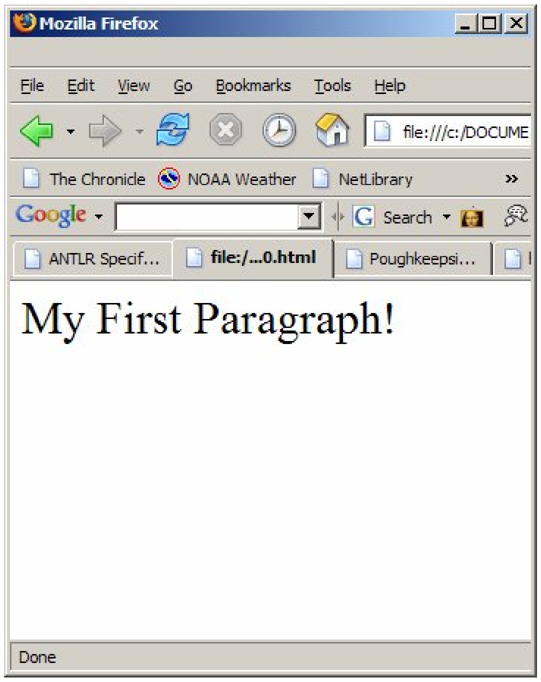
Figure 4: HGL Output: HTML of myHGLPage.hgl

2.1.3 Generate Your Web Page

Run your HGL source to generate your web page:

```
$ java hgl -omyHGLPage.html myHGLPage.hgl
```

The above execution will produce the file `myHGLPage.html`. When opened in a web browser, it will look like Figure 5.



My First Paragraph!

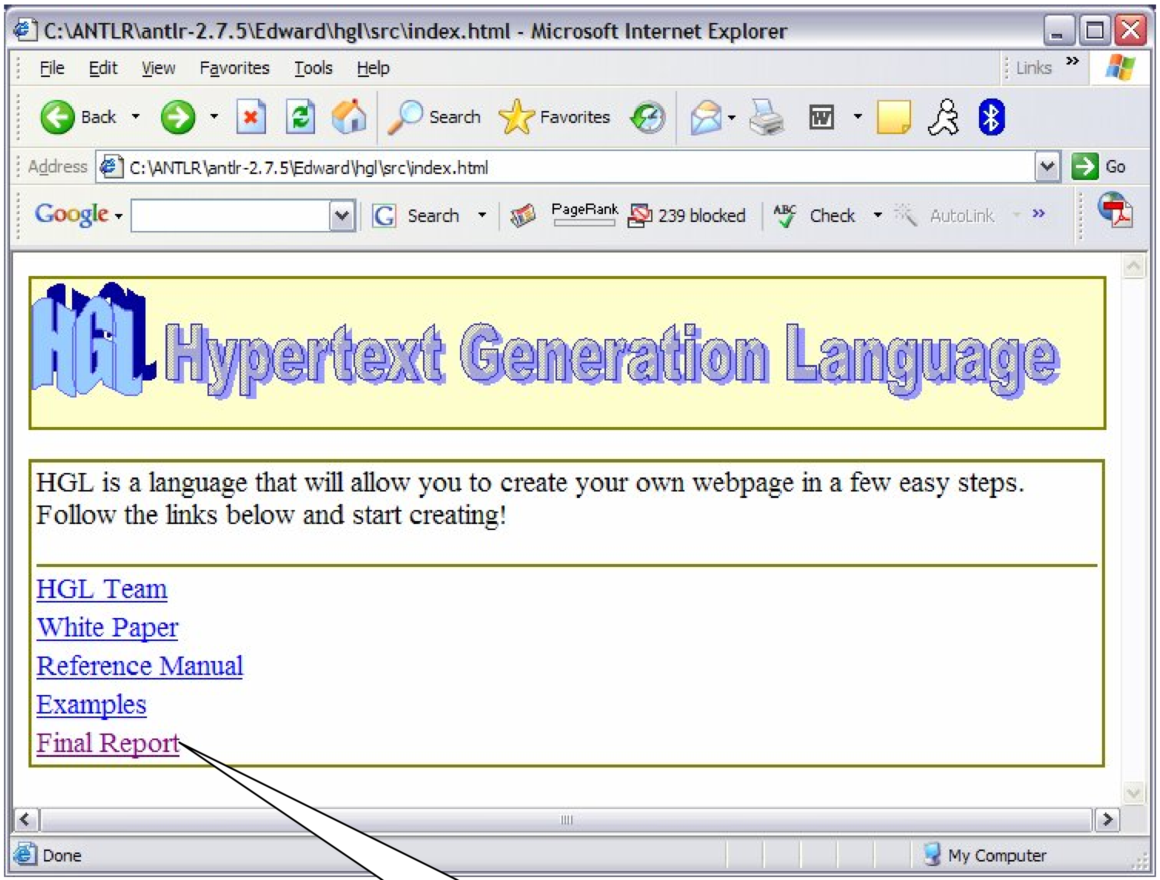
Figure 5: A simple example

2.2 *Advanced Examples*

2.2.1 Tables, Links, and Images

In this section of the tutorial, examples that use other features of HGL, like tables, links and images, will be described.

Figure 6 shows an example of an HGL generated web page with tables, hyperlinks and images. The HGL code for this screenshot is show in Figure 7. Usage of Links is shown in line 29: the "addr" property sets the URL the link points to, while the add() operation on Link objects is the text, or image, that will be clickable. The set() routine is a synonym for the 'obj.property="value"' property setting syntax; it takes two arguments, the property and value, both as strings. Table is usage is shown in lines 43-48. The addElement() routine must be used to add items to tables. The first item is an object that obeys the same rules as the the argument to add(), and the next two are expressions that must evaluate to an integer specifying the row and column. Items do not have to be added to consecutive rows/columns. HGL will construct the correct sized sparse table if entries are added out-of-sequence, adding empty cells as needed.

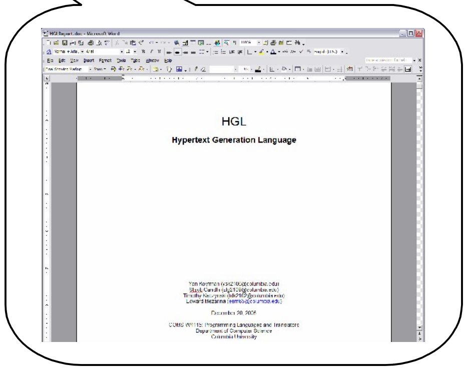

C:\ANTLR\antlr-2.7.5\Edward\hgl\src\index.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\ANTLR\antlr-2.7.5\Edward\hgl\src\index.html
Done
My Computer

HGL Hypertext Generation Language

HGL is a language that will allow you to create your own webpage in a few easy steps. Follow the links below and start creating!

HGL Team
White Paper
Reference Manual
Examples
Final Report

Figure 6: An example with Links and Tables.



HGL

Hypertext Generation Language

[illegible]

December 20, 2006

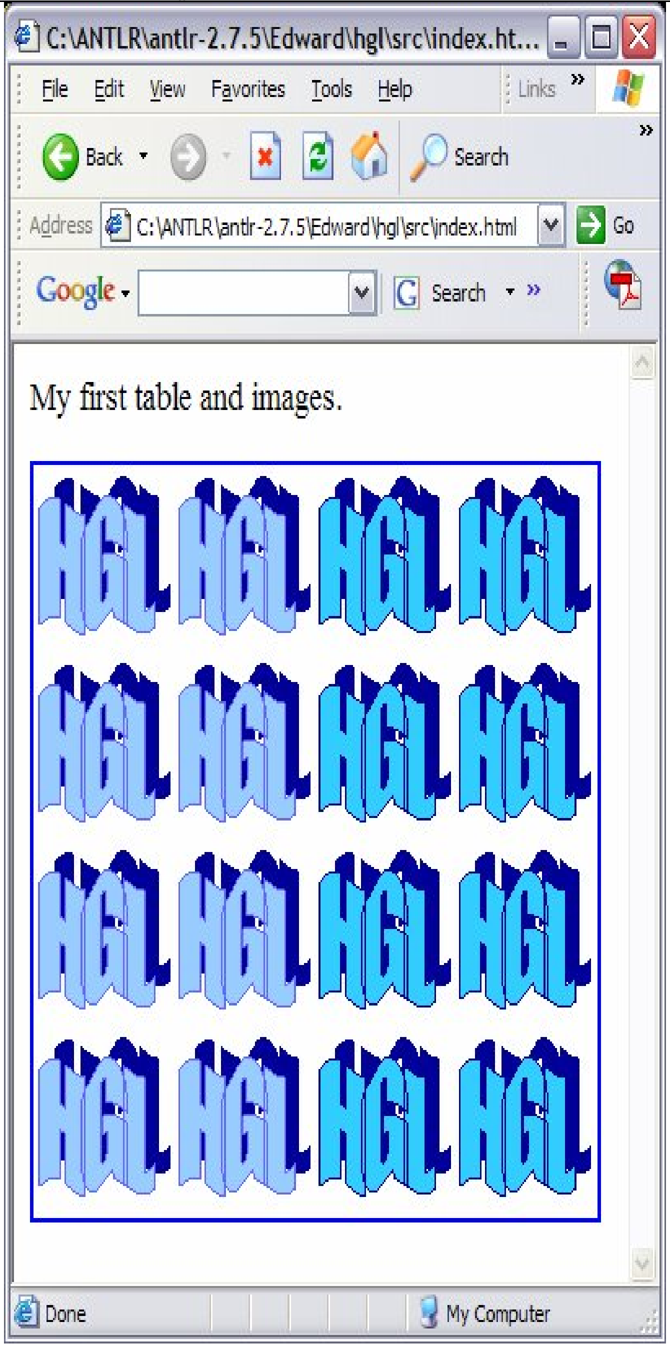
COMS W4115: Programming Languages and Translators
Department of Computer Science
Columbia University

```
//---------------------------------
// hglweb.hgl
//---------------------------------
// Variable Declarations
Paragraph p1; Paragraph p2;
Paragraph p3; Paragraph p4;
Paragraph p5; Paragraph p6;
Paragraph p7;
Image img1;   Image img2;
Link l1; Link l2; Link l3;
Link l4; Link l5; Link l6;
Page p;  Table t;

// Variable initialization
$img1.setPath("hgl2.gif");
$img2.setPath("hglLong.gif");
$t.set("border","thin solid olive");

// Pane initialization
$p1.set("bgcolor", "#ffffcc");
$p1.set("border", "thin solid olive");
$p1.set("width", 700 | "px");
$p1.set("height", 100 | "px");
$p1.add($img1); $p1.add($img2);


// Setting Links
$p2.add("HGL is a language that will allow you to create
    a.  your own  webpage  in a few easy steps.
    b.  Follow the links below and start  creating!");
$p2.newline(2);
$p2.set("border-bottom","thin solid olive");


$p3.add("HGL Team");
$l1.add($p3);
$l1.set("addr","C:\\ANTLR\\antlr-2.7.5\\hgl\\src\\team.gif");

$p4.add("White Paper");
$l2.set("addr","C:\\ANTLR\\antlr-2.7.5\\hgl\\src\\whitePaper.doc");
$l2.add($p4);

$p5.add("Reference Manual");
$l3.set("addr","C:\\ANTLR\\antlr-2.7.5\\hgl\\src\\refMan.doc");
$l3.add($p5);

$p6.add("Examples");
$l4.set("addr","C:\\ANTLR\\antlr-2.7.5\\hgl\\src\\examples.doc");
$l4.add($p6);

$p7.add("Final Report");
$l5.set("addr","C:\\ANTLR\\antlr-2.7.5\\hgl\\src\\HGLReport.doc");
$l5.add($p7);

// Add Elements to table.
$t.addElement($p2, 0, 0);
$t.addElement($l1, 1, 0);
$t.addElement($l2, 2, 0);
$t.addElement($l3, 3, 0);
$t.addElement($l4, 4, 0);
$t.addElement($l5, 5, 0);

// Adding elements to page.
$p.add($p1);
$p.add($t);

// Print page.
$p.print();
```

Figure 7: HGL Code for Figure 6

2.2.2 Functions/Table/Loops/Conditionals

Figure 8, below, shows the use of images, functions and more use of tables. Lines 13 and 15 show the setPath() routine, used to set the location of the image to be displayed. Lines 23-40 show a function definition that takes a Table object as an argument. HGL function arguments are passed-by-reference, allowing routines like the one shown here to modify their argument. Lines 26-38 show usage of the while construct. This is the only looping primitive available in HGL; the index variables must be incremented at the bottom of each loop body. The variables declared inside fillTable() are not visible outside that function, since a function block defines a new scope.


C:\ANTLR\antlr-2.7.5\Edward\hgl\src\index.ht...
File Edit View Favorites Tools Help
Links
Back
Search
Address C:\ANTLR\antlr-2.7.5\Edward\hgl\src\index.html
Go
Google
Search
Done
My Computer

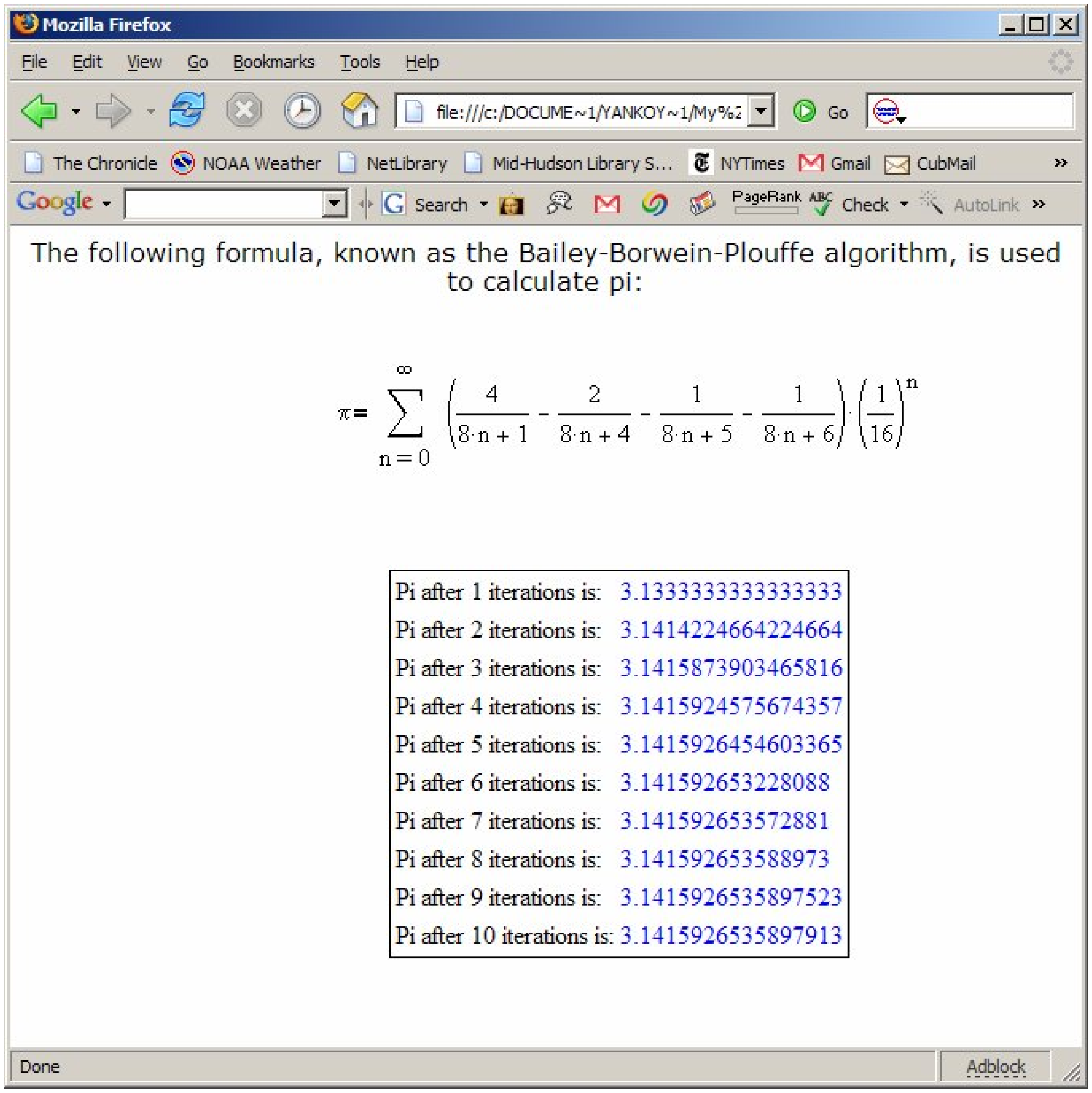
My first table and images.

Figure 8: Output and source for sample2.hgl.

```
1   //-------------------------------------
2   // sample2.hgl
3   //-------------------------------------
4   int i; int j;
5   Paragraph pr;
6   Page p;
7   Table t;
8   string url;

9   $p.add($pr);
10  $pr.add("My first table and images.");

11  $t.set("border","thin double blue");

12  Image img1;
13  $img1.setPath("hgl2.gif");
14  Image img2;
15  $img2.setPath("hgl.gif");

16  fillTable($t);

17  $p.add($t);
18  $p.print();

19  //-------------------------------------
20  // Function: fillTable
21  // Fills a table's content with data
22  //-------------------------------------
23  int fillTable(Table tbl) {
24    int r; int c;
25    $r = 0; $c = 0;

26    while($r < 4) {
27      $c=0;
28      while($c < 4) {
29        if ($c < 2) {
30         $tbl.addElement($img1, $r, $c);
31        }
32        else {
33         $tbl.addElement($img2, $r, $c);
34        }
35        $c = $c+1;
36      }
37      $r=$r+1;
38    }
39    return 0;
40  }
```

2.2.3 Arithmetic Operations in HGL

Figure 9 and Figure 10 show further use of tables, functions and arithmetic. This shows an example of a feature of HGL not available with plain HTML: arithmetic. Here, it is being used to calculate pi. New features shown here are the link() function, on line 58, and the string concatenation operator, on line 61. The link() routine enables multiple objects to share the same formatting. In this case, it is used to set the value of pi with the blue foreground color. Once objects are linked, they share the same CSS class. This helps reduce the size of the resulting HTML file as well as to eliminate the need to set identical properties on several different objects. The string concatenation operator on line 61 is the pipe, or "|" character. Its arguments can either be a string or an expression returning a string or integer. Here it is used to add the iteration index to the first column on in the table.



The following formula, known as the Bailey-Borwein-Plouffe algorithm, is used to calculate pi:

$$\pi = \sum_{n=0}^{\infty} \left(\frac{4}{8 \cdot n + 1} - \frac{2}{8 \cdot n + 4} - \frac{1}{8 \cdot n + 5} - \frac{1}{8 \cdot n + 6} \right) \cdot \left(\frac{1}{16} \right)^{n}$$

Pi after 1 iterations is:	3.1333333333333333
Pi after 2 iterations is:	3.1414224664224664
Pi after 3 iterations is:	3.1415873903465816
Pi after 4 iterations is:	3.1415924575674357
Pi after 5 iterations is:	3.1415926454603365
Pi after 6 iterations is:	3.141592653228088
Pi after 7 iterations is:	3.141592653572881
Pi after 8 iterations is:	3.141592653588973
Pi after 9 iterations is:	3.1415926535897523
Pi after 10 iterations is:	3.1415926535897913

Figure 9: Table and Image used

```
float pi(int n)
{
  float x;
  $x  = ((4.0/(8.0*$n+1.0)) - (2.0/(8.0*$n+4.0)) -
         (1.0/(8.0*$n+5.0)) - (1.0/(8.0*$n+6.0)));

  float z;
  if ($n == 0)
  {
    $z = 1.0;
  }
  else
  {
    int y;
    $y = 1;

    $z = 1.0 / 16.0;

    while ($y < $n)
    {
      $z = $z * (1.0 / 16.0);
      $y = $y + 1;
    }
  }
  return $x * $z;
}
Paragraph para;
$para.add("The following formula, known as the Bailey-Borwein-Plouffe"
           | " algorithm, is used to calculate pi:");
$para.text-align = "center";
$para.font-size = "12pt";
$para.font-family = " ""Verdana"" ""sans-serif"" ";
Image piImg;
$piImg.src = "http://www.geocities.com/SiliconValley/Bay/9187/plffe1.gif";
$piImg.left = "30%";
$piImg.top = "20px";
$piImg.position = "relative";
float p;
$p = 0.0;
int x;
$x = 0;
// Number of iterations
int iter;
$iter = 10;
// The table
Table tbl;
$tbl.inherit($piImg);
$tbl.border = "thin solid black";
$tbl.left = "35%";
$tbl.top = "80px";
Paragraph pi[$iter];
while ($x < $iter)
{
  $p = $p + pi($x);
  if ($x==0)
 { $pi[$x].fgcolor = "blue"; }
  else
 { $pi[$x].link($pi[0]); }
  $pi[$x].add($p);

  $tbl.addElement("Pi after " | $x + 1 | " iterations is: ", $x, 0);
  $tbl.addElement($pi[$x], $x, 1);
  $x = $x + 1;
}
Page pg;
$pg.add($para);
$pg.add($piImg);
$pg.add($tbl);
$pg.print();
```

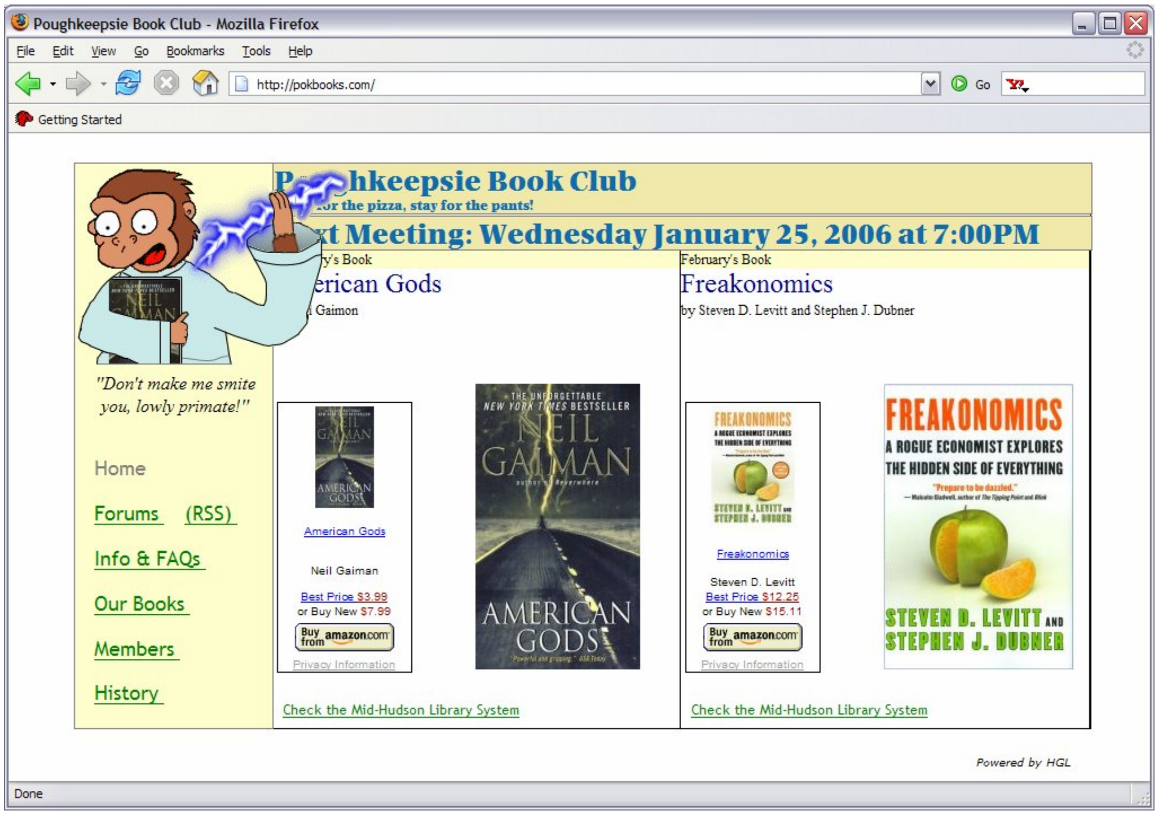
Figure 10: Code for algorithm in Figure 9

2.2.4 Real-Life Example

In Figure 11, below, shows an example of a real, production Web site (www.pokbooks.com), for which one of us is a web-master, that was created with HGL. Everything except for the graphics themselves were generated from HGL source that totaled about 400 lines of code. The HGL interpreter produced an HTML file (combined HTML and CSS) of about 8kb. The source code for this example is shown in Appendix B.

Several features HGL made creating this page easier than it might have been with plain HTML. This includes using arrays and the link() function to create a series of Link objects with the same formatting (Figure 13, in red), and use of the related inherit() function (Figure 15, in red), for copying properties from a similar object, yet still retaining the ability to independently change them. Linking produces CSS and HTML as excerpted in Figure 12: the objects share only one CSS class and their "class" attribute links to that one class. Note that linking does not mean they share "href" attributes: non-CSS attributes are not shared and can be set independently.

But perhaps the most helpful aspect of programming in HGL instead of HTML is its ability to perform arithmetic operations. The trick to this web site is absolute positioning: it was divided into several rectangular regions, including a navigation pane, title areas, and a center area for the main content. To produce a layout where all regions fit together neatly, each area's length, width and location on the page was specified in absolute coordinates generally using the number of pixels from a certain edge of the screen (for example, the .left property shown in Figure 15 in blue). Near the beginning of the program, variables (in effect, constants) to define a few basic dimensions were created (Figure 14) and were then used throughout the rest of the program. This limits the number of hard-coded numbers needed in the rest of the program, and makes it easy to resize the entire web site by simply changing one of the constants and recompiling.



P hkeepsie Book Club
or the pizza, stay for the pants!
xt Meeting: Wednesday January 25, 2006 at 7:00PM

"Don't make me smite you, lowly primate!"

Home
Forums (RSS)
Info & FAQs
Our Books
Members
History

y's Book
erican Gods
Gaimon

American Gods
Neil Gaiman
Best Price $3.99
or Buy New $7.99
Buy from amazon.com
Privacy Information

Check the Mid-Hudson Library System

February's Book
Freakonomics
by Steven D. Levitt and Stephen J. Dubner

Freakonomics
Steven D. Levitt
Best Price $12.25
or Buy New $15.11
Buy from amazon.com
Privacy Information

Check the Mid-Hudson Library System

Powered by HGL

Figure 11: See Appendix C for complete HGL source code, under the heading "pbc.hgl".

```
...

// Set up format for links
$idx = 0;
$navLinks[$idx].margin-left = "10%";
$navLinks[$idx].line-height = "1.1";
$navLinks[$idx].background =
"transparent";
$navLinks[$idx].fgcolor = "green";

...

while ( $idx < arraySize($navLinks) )
{
  $navLinks[$idx].link($navLinks[0]);
  $idx=$idx+1;
}

...
```

Figure 13: Linking of objects

```
a.Link0
{
  font-size: 110%;
  color: green;
  vertical-align: top;
  background: transparent;
  line-height: 1.1;
  font-family: "Trebuchet MS", "Avant Garde", sans-serif;
  margin-left: 10%;
}

...

      <a class="Link0" href="phpBB2/rss.php">
        (RSS)
      </a>

      <br>
      <br>
      <a class="Link0" href="faq.html">
        Info & FAQs
      </a>
...
```

Figure 12: Portion of HTML genarated from code in Figure 13

```
// Top margin for all boxes
int topMargin;
$topMargin = 10;

// Height (in pixels) of main page constructs
int mainHeight;
$mainHeight = 500;

// Left margin (in pixels) of main box
int mainLeft;
$mainLeft = 50;

// Width(in pixels) of main box
int mainWidth;
$mainWidth = 900;
```

Figure 14: Specifying the dimensions of the main containing block.

```
// Title
Paragraph nextBookTitle;
$nextBookTitle.inherit($currBookTitle);
$nextBookTitle.left = $bkWidth + $mainLeft + $navWidth | "px";
$nextBookTitle.border = "0 none transparent";
$nextBookTitle.border-left = "thin solid black";
$nextBookTitle.add("Freakonomics");
$nextBookBox.add($nextBookTitle);
```

Figure 15: Inherit copies properties from another object, and using HGL arithmetic to calculate dimensions.

3.0 Language Reference Manual

3.1 Grammar Notation

The reference manual for the Hypertext Generation Language follows. Text in this typeface implies an example of HGL syntax as it would appear in a source file, while text in standard and slanted typefaces denotes explanatory passages.

3.1.1 Minimum File

An HGL file must consist of at least on statement or declaration, as they are defined later in this section. An empty file is not a valid HGL source file.

3.2 Lexical Conventions

3.2.1 Comments

Comments are prefixed with two forward slash characters //, which instruct the compiler to ignore the remainder of the line.

3.2.2 Whitespace

Whitespace separates tokens in HGL, and includes space, tabs and new lines. The amount of whitespace between elements is not important except when it appears in a string.

3.2.3 Identifiers

Identifiers consists of letters, digits, and underscore. The first character of the identifier should be a non-digit or an underscore. Upper and lower case letters are considered different.

3.2.4 Keywords

These identifiers are reserved as keywords and may not be used in other contexts where they are not explicitly allowed by this document:

```
boolean   while
paragraph Link
char      OList
else      UList
if        Page
int       true
string    false
continue  break
return    Table
Image     Span
```

Functions that are part of the HGL library also represent a reserved word. This includes:

```
arraySize()
```

3.2.5 Strings

Strings are sequences of characters enclosed by double quotes "abc". A double quote inside the string is escaped by a another double quote "".

3.2.6 Operators

The following arithmetic operators are permitted for integer and float types:

+	Addition
-	Subtraction
/	Division
*	Multiplication
=	Assignment (valid for string and boolean objects also)
- / +	Unary plus / minus

Boolean operators, applicable to integer, float and boolean types, include:

<	Less than
>	Greater than
<=	Less than or equals
>=	Greater than or equals
==	Equality
!=	Inequality
||	Boolean or
&&	Boolean and

The following non-arithmetic operators are permitted:

. (dot)	HTML Object function access
|	String concatenation

3.2.7 Other Tokens

Some symbolic characters or sequences of symbolic characters are used in the language:

{ }	Group statement blocks
( )	Enclose function arguments
[ ]	Indicate array size when appended to variable declaration, array initializers
,	Separate elements in lists
;	Statement terminator
$	When followed by a keyword, denote variable names

3.2.7.1 Variables

Variables are represented by a dollar sign, “$”, followed by the identifier, or name, of the variable.

Variable names are case-sensitive. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores, the same rules as for identifiers.

3.2.8 Types

3.2.8.1 Available Types

HGL supports a subset of standard data types along with this language's built-in Data Type Objects representing HTML constructs (on gray background).

Type	Declaration Syntax	Sample Values
String	`string` *`var;`*	"My_html_page1"
Integer	`int` *`var;`*	-2147483648 to 2147483647
Float	`float` *`var;`*	3.4e+05 (see below)
Boolean	`boolean` *`var;`*	`true` or `false`
Page	`Page` *`var;`*	Opaque type**
Paragraph	`Paragraph` *`var;`*	Opaque type
Table	`Table` *`var;`*	Opaque type
Image	`Image` *`var;`*	Opaque type
Link	`Link` *`var;`*	Opaque type
Span	`Span` *`var;`*	Opaque type
Unordered List	`UList` *`var;`*	Opaque type
Ordered List	`OList` *`var;`*	Opaque type
Array	*`type var[int];`*	[positive integer]

*A floating point number consists of an integer part, a decimal point, a fraction part and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. The decimal point and fraction part, and the ('e'|'E') and exponent are optional.

**Opaque types are HTML objects that can only be accessed and modified through member functions, accessed via the dot (.) operator. Details can be found in HGL Library on page 25.

3.2.8.2 Opaque Types

Though they follow the same naming and instantiation rules as normal primitive types, the opaque objects representing HTML constructs are typically modified through member functions and attributes. These objects are described in the HGL Library section.

3.2.9 Expressions

3.2.9.1 Literal Constants

Literal constants include all references to int, float, string, or boolean values which appear in program code as a literals and not stored in variable.

3.2.9.2 Parenthesized Expressions

An expression enclosed in parenthesis () has the same value as the identical expression written without parenthesis. Parenthesis can be nested to an arbitrary degree, e.g., (((exp))) provided parenthesis are balanced.

3.2.9.3 Function Calls

Function call syntax is similar to C++. Two kinds of functions exist:

1. User defined, global functions, not associated with objects. Users can define new functions of this kind. examples:

```
$var = fun();                functions returning values
do_work(10, "a string");     example of argument list
```

2. HTML object functions. These cannot be redefined by users, and new HTML object functions cannot be added from within HGL. Examples:

```
$page.print();          void functions not returning values
$page.add("text");
```

3.2.9.4 Arithmetic Expressions

HGL supports four arithmetic operations for integers and floating point numbers: + - / *. See below for precedence rules.

3.2.9.5 Operators

HGL supports a set of relational and conditional operators that will allow the user to determine the relationship between two values. For this purpose, HGL will allow the use of relational and conditional operators in conjunction to construct more complex decision-making expressions.

3.2.9.6 Relational

Operator	Use	Description
`>`	`v1 > v2`	Returns `true` if `v1` is greater than `v2`
`>=`	`v1 >= v2`	Returns `true` if `v1` is greater than or equal to `v2`
`<`	`v1 < v2`	Returns `true` if `v1` is less than `v2`
`<=`	`v1 <= v2`	Returns `true` if `v1` is less than or equal to `v2`
`==`	`v1 == v2`	Returns `true` if `v1` and `v2` are equal
`!=`	`v1 != v2`	Returns `true` if `v1` and `v2` are not equal

3.2.9.7 Conditional

Operator	Use	Description
`&&`	`V1 && v2`	Returns `true` if `v1` and `v2` are both `true`; conditionally evaluates `v2`
`||`	`V1 || v2`	Returns `true` if either `v1` or `v2` is `true`; conditionally evaluates `v2`
`!`	`!vx`	Returns true if vx is false

3.2.10.8 Operator Precedence

Precedence rules and associativity, from highest to lowest.

Operator	Associativity
`[]`	`Left`
`.`	`Left`
`+ - (unary)`	`Right`
`* /`	`Left`
`+ -`	`Left`
`< <= > >=`	`Non-associative`
`== !=`	`Non-associative`
`&&`	`Left`
`||`	`Left`
`=`	`Right`

3.2.10 Statements

HGL statements are composed of either simple expressions terminated by a semicolon, compound statements surround by { and } and the control flow and iteration statements described below. Statements are executed in order from the beginning of the file to the end, unless explicitly redirected through function calls, iteration or control-flow structures. Execution of statements starts with the first line in the file, not from any specially named function.

3.2.10.1 Array Initialization

Arrays of simple objects, which includes int, boolean, and string and floats, can be initialized after arrays are created using the [val1, val2, ...] syntax. Example:

```
int vals[3];
vals = [9,6,3];
```

3.2.10.2 Declarative Statements

Variables All variables must be declared before they are used. Declaration syntax is similar to that of C, except that variables cannot be initialized at the time of their introduction. Storage space is reserved for variables upon their declaration. Expressins can be used in place of integer literals to describe the size of an array upon declaration.

Examples:

```
int color;
Page data[5];
```

```
ULList lists[$var+10];
```

Functions All functions must be defined before they are used, but function declaration syntax allows function names to be introduced and types and arguments declared before the functions themselves are defined, as in C or C++. Function blocks are delimited by braces.
The following example shows a function being declared, and later defined:

```
int foo(string s);
arbitrary code
int foo(string s) { statements }
```

Neither function declarations nor function definitions may be nested.

Scope Names are available in global scope or block scope. A block is delimited by curly braces, { and }. Variables declared in global scope are visible globally and in all functions; variables defined within a block are only visible in that block, and override the values of similarly named variables in parent blocks.

3.2.10.3 Control Flow Statements

Statement Type	**Keyword**	**Example**
`Iterative`	`while`	while (condition) {statement};

`Condition`	`if-else`	if (condition) {statement};
		
		if (condition) {statement}
		else {statement};

break	- *Break out of a while loop or if statement.*
continue	- *Skip to next iteration of while loop.*
return *expression*	- *Leave function, optionally returning a value.*

3.3 HGL Library

3.3.1 Built-in HTML Objects and Properties

The following sub-sections describe HTML constructs and properties that are available in the language.

3.3.1.1 Common CSS Properties

The following are CSS properties supported by HGL. See Appendix A on page 46 for more details and examples. They can be set on any HTML object through either of the object.set(“property_name”, “property_value”) or object.property_name = “property_value” syntax. Note that HGL does not check for semantic validity of a CSS property value to a given property. For example, if the fgcolor property is set to “large”, HGL will not issue a warning. However. attempts to set a property not from the below list will be trapped and a warning issued. Running the HGL translator with the “–p” flag prints all available properties for every type.

• font-size:	font size of object (i.e. 12pt, large)
• font-style:	One of: normal, italic, or oblique
• font-family:	Font family style (i.e. serif, sans-serif)
• font:	catch-all property for all font properties (i.e. italic, bold, serif)
• text-align:	Aligns a block of text (i.e. center, justify)
• width:	Assigns a width to an element (i.e. 10em, 30)
• height:	Assigns a height to an element (i.e. 10em, 30)
• float:	Allows to wrap text around an element
• border:	border around object (i.e. solid, blue)
• border-left:	Sets width, style, and color for an elements left border
• border-right:	Sets width, style, and color for an elements right border
• border-top:	Sets width, style, and color for an elements top border
• border-bottom:	Sets width, style, and color for an elements bottom border
• background:	shorthand for background property (i.e. #7fffd4, white)
• fgcolor:	text (“foreground”) color (i.e. #000079, blue)
• bgcolor:	background color (i.e. #000080, white)
• bgimage:	background image (i.e. url: http://hgl/mypicute.gif)
• line-height:	Controls spacing between baseline of test (i.e. 200%)
• z-index:	Relative order in the z-dimension of the page.
• margin:	The left margin of an element (i.e. 0, 20)
• margin-left:	The left margin of an element (i.e. 0, 20)
• margin-right:	The right margin of an element (i.e. 0, 20)
• margin-bottom:	The bottom margin of an element (i.e. 0, 20)
• margin-top:	The top margin of an element (i.e. 0, 20)
• padding:	Property short hand for ones below.
• padding-bottom:	Space between bottom border and element
• padding-left:	Space between left border and element
• padding-right:	Space between right border and element
• padding-top:	Space between top border and element
• position:	Relative, absolute or auto: how lengths are interpreted
• top:	Offset between top margin edge and top of containing block
• bottom:	Offset between bottom margin edge and bottom of containing block

• left:	Offset between left margin edge and left of containing block
• right:	Offset between right margin edge and right of containing block
• vertical-align:	Aligns an element vertically (i.e. middle, baseline)

3.3.1.2 Common Functions

3.3.1.2.1 Common HTML Object Methods

These functions are availble on HTML objects through the “obj.<function>” syntax.

- `set(string` *`name`*`, string/number` *`value`*`)`
  - Sets the CSS property *name* to *value*. Synonym for “object.property = value” syntax.
- `setPseudo(string` *`pc_name`*`, string` *`prop_name`*`, string/number` *`value);`*
  - Set pseudo-class pc_name’s property prop_name to value. The valid pseudo-classes for all HTML objects are focus, hover, and active.
- `inherit(`*`HTML Object`*` o)`
  - Copies all CSS properties from *o* to current object.
- `link(`*`HTML Object`*` o)`
  - Similar to inherit(), but a stronger link: any changes made to properties in this object will be reflected in the properties of the linked object. This feature provides CSS class sharing.
- `newline(integer` *`num_lines`*`)`
  - `In`sert *num_lines* blank lines into this object’s string representation. Will be replaced in the HTML output by a corresponding number of <br> tags.

3.3.1.2.2 Non-HTML Functions

- `int arraySize(array` *`a`*`)`
  - Returns the size of an array.

3.3.1.3 Unique HTML Object properties and functions

3.3.1.3.1 Page

The root container of all other objects. Adding items to a Page object and then printing it out is the only way to obtain output from HGL.

Functions
- `print()`
  - output HTML/CSS code for this Page object, including any objects that have been *add*ed to it.
- `add(any type` *`obj`*`)`
  - Adds an object to a page. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.
- `setTitle(string` *`name`*`)`
  - Set this Web page’s title.

3.3.1.3.2 Paragraph

The basic block-level HTML element. It can contain any data.

Functions

- `add(any type obj)`
    - Adds an object to a paragraph. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.

3.3.1.3.3 UList (unordered list)

A simple bulleted list. Because of nesting, lists can contain other lists.

Functions

- `addItem(any type obj)`
    - Adds an object to the list. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.

3.3.1.3.4 OList (ordered list)

A numbered list.

Functions

- `addItem(any type obj)`
    - Adds an object to the list. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.

3.3.1.3.5 Table

Tables can be of any size. Blank cells are added as needed to create a rectangular table given the locations of the items that have been added with addElement().

Functions

- `addElement(any type obj, int row,` int *column*`)`
    - Adds an object to the table in a particular row and column. This element can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.

3.3.1.3.6 Link

HTML anchor, this type displays its objects between `<a>` and `</a>` tags. Use the special HGL attribute addr to set the "href" property, the destination address of the link.

Functions

- `add(any type obj)`
    - Adds an object to a link. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page. This object will be highlighted and clickable.

Properties

- `addr`
    - The URL to which this link points.
- link

    - A pseudo-class, set with the setPseudo function. The object will take on the link pseudo-class's attributes if it has not been visited.

- visited
    - A pseudo-class, set with the setPseudo function. The object will take on the visited pseudo-class's attributes if it has already been visited.

3.3.1.3.7 Span

Objects added to this type are inserted between the `<span>` and `</span>` inline element. Useful for changing the formatting of inline elements when a wrapping text in a containing paragraph would not produce the desired result.

Functions

- `add(any type obj)`
    - Adds an object to a span. This can be can HTML object, non-HTML primitive or expression valid in HGL, except for Page.

3.3.1.3.8 Image

1. Functions:

- `setPath(string location)`
    - Sets the path where the image will be obtained from.

4.0 Project Plan

4.1 Process used

4.1.1 Planning and specification

For planning and specification purposes a series of meetings were held to first elect a team leader, decide the objectives/responsibilities of the group, type of development process, and last, decide on tools necessary to keep track of the team's progress and commitments to stated deadlines.

The programming style model used for this project is the Incremental Model, which allowed us to construct the software step by step. In this sense, HGL was designed, implemented, integrated, and tested as a series of incremental steps starting with the project proposal, specification, architectural design, HGL Scanner, HGL parser, HGL tree walker, HGL Logger, and HGL Objects.

The main tool used for management purposes is dotProject, which is a "PHP web-based Management framework that includes modules for companies, projects, tasks (with Gantt charts), forums, files, calendar, contacts, tickets/helpdesk, multi-language support, user/module permissions and themes"[1].

4.1.2 Development and Testing

For development and testing practices, we relied on the programming style described in 4.3 Programming Style.

The tool used for these two processes as well as in the specification phase was CVS. This tool provided the team with the ability to merge, compare, and version control the files in the repository.

4.2 Team Responsibilities

Refer to timeline section 4.4 for details on team responsibilities.

4.3 Programming Style

For development and testing, the team dynamics used is similar to Pair Programming, in which code is written by sharing a single computer in most cases, except we usually had all four group members sitting together. This approach not only allowed the team members to conceptualize and visualize the design and goals of the project in unison, but also allowed team members to share knowledge among them. Lastly, it allowed for faster programming because errors and/or potential pit falls were caught during the development process. This team approach was especially beneficial during HGL's front end development.

4.4 Project Timeline

The table below describes team responsibilities and timelines established for the development of HGL.

Task Name	Task Description	Assigned To	Task Start Date	Task End Date	Completion
Scanner	Add Front End scanner support for HGL classes.	Yan Koyfman, Tim Kaczynski, Edward Mezarina, Shruti Gandhi	09/25/2005	10/11/2005	100%
Parser	Add Front End Parser support for semantic analysis of HGL classes.	Yan Koyfman, Tim Kaczynski, Edward Mezarina, Shruti Gandhi	10/13/2005	10/25/2005	100%
Tree generating code	Modify parser such that ANTLR generates an AST with the designed structure.	Yan Koyfman, Tim Kaczynski, Edward Mezarina, Shruti Gandhi	10/18/2005	10/31/2005	100%
Tree Walker	Add base tree walker support	Yan Koyfman, Shruti Gandhi, Edward Mezarina, Tim Kaczynski	10/21/2005	10/31/2005	100%
Symbol table	Add scoping support through symbol tables.	Timothy Kaczynski	10/31/2005	11/14/2005	100%
Logging	Add logging support for debugging purposes.	Shruti Gandhi, Yan Koyfman	11/18/2005	11/22/2005	100%
Regression test	Create test cases for each HGL feature and report success/failure.	Shruti Gandhi, Yan Koyfman	12/01/2005	12/20/2005	100%
Line numbers in errors	Add line numbers to the logger infrastructure.	Shruti Gandhi, Yan Koyfman	11/18/2005	11/29/2005	100%
HTML Objects	Add support for HTML objects(Page, Paragraph, Link, etc), and CSS related properties (Font, Border,etc.)	Yan Koyfman	10/31/2005	12/18/2005	100%

4.5 Software Project Environment

4.5.1 Software

The software used to develop HGL were:

- ANTLR (Version 2.4).
- JAVA (JDK 1.4).

4.5.2 Tools

- CVS: Developers version control.
- ViewCVS: CVS Browser

For file management we ran CVS and ViewCVS, which provided Web-based access to the CVS repository and allowed simple browsing of diffs and progress.

- dotProject: Project management application.

dotProject benefited us in a couple of ways. First, just having a central location that is accessible to all members to enumerate the myriad tasks that go into a project was useful, as well as a single place where repository for comments related to individual tasks and defect tracking. Its time-tracking features were less used; as developers we knew from experience that adhering to a schedule dreamed up at the start of a project is close to impossible. Task progress and end-dates required the flexibility software could not provide.

4.6 Project Log

The table below shows the timeline of the HGL development process from the conception this language to its final version. The source for this table comes from our CVS Logs, DotProject, and meeting minutes.

Milestones	Timeline
Conception of HGL	**September 25, 2005**
Proposal (white paper) completed	**September 26, 2005**
HGL Scanner work in progress	**October 04, 2005**
LRM work begins	**October 07, 2005**
First HGL grammar unit test	**October 11, 2005**
Parser Work begins	**October 13, 2005**
Completed LRM	**October 18, 2005**
Tree Generation and Functions work	**October 19, 2005**
Tree walker work completed and tested	**October 31, 2005**
Symbol table support added	**November 14, 2005**
HTML Objects work begins	**November 18, 2005**
Logging and Error handling work begins	**November 18, 2005**
Logging and Error handling work complete	**November 29, 2005**
Function execution support complete	**December 06, 2005**
HTML Object work ends	**December 17, 2005**
Completed all Development	**December 18, 2005**
Completed final regression test	**December 19, 2005**
Final Report	**December 20, 2005**

5.0 Architecture Design

There are three main components that make up HGL: back end, which is primarily responsible for generating HTML, the front end, which handles syntactic analysis of the input, and the logger and exception classes. Figure 16 shows the execution path and data flow among the components. The interfaces between sub-components in figured are labeled 1-7 for data flow and E and L for out-of-band output for logging and error handling:

1. A source file is read in by the compiler and input into the lexer.
2. The lexer scans the file and generates tokens that are fed into the parser.
3. The parser uses HGL language syntax rules to build tree nodes.
4. The tree walker interprets the tree nodes and carries out operations with calls to back end classes to compute expressions and handle HTML operations.
5. The symbol table keeps track of variables and functions declared in the code.
6. The tree walker eventually dispatches function calls to the HTML object handling code.
7. The HTML handling code has access to hash tables with CSS properties and pseudo-class properties and converts the tree nodes to HTML.
8. When the print() function on Page is called, the contained objects are requested to output their HTML representation. At this point indentation is added to the output based on objects' nesting level within the page.

E: The red lines indicate exceptions being thrown. There are several exception classes, some specialized to contain line number information, and subclasses to indicate the severity of the error.

L: The HGLCommon class contains a Java Logger object that is shared by all other components of the translator.

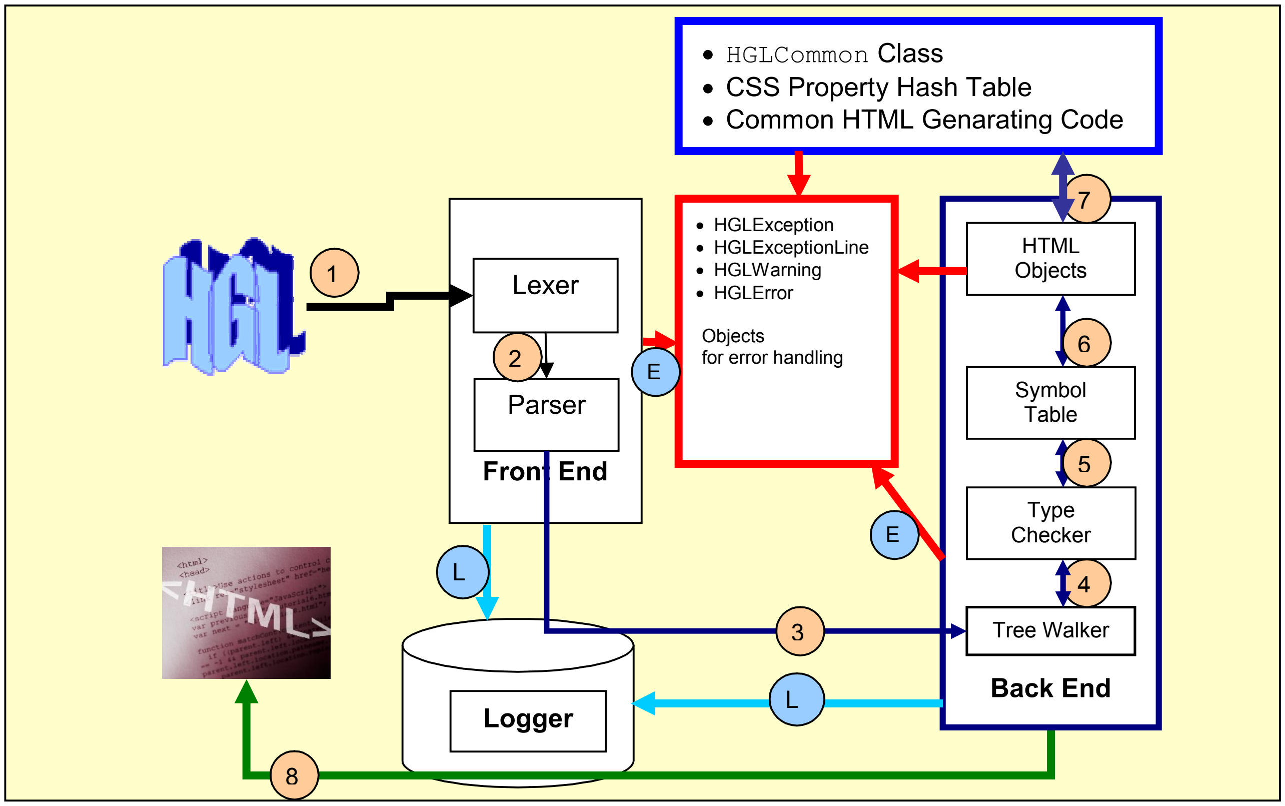


Figure 16: Data flow in HGL translator showing front-end and back-end paths.

5.1 Front End

The front end consists of the lexer and parser, which were generated from an ANTLR grammar. Automatic tree building was enabled, though for some rules the trees were re-written for easier handling in the tree walker.

The front end components were implemented by the entire group in joint meeting sessions.

5.2 Back End

The back end includes the tree walker and all HTML generating code. The tree walker handles tasks such as:

1. Expressions: returns object
   a. arithmetic expressions
   b. function calls
   c. logical expression evaluation
2. Statements
   a. Declarations: functions added to symbol table (first pass)
   b. Declarations: variables added to symbol table (second pass)
   c. dispatch HTML function via HGLCommon
   d. While loops
   e. Alternation – if/else statements

Two passes are made over the source code by the tree walker. This first pass is dedicated solely to picking up function declarations and definitions. This allows for code that calls functions defined after the code, as well as functions that call each other.

The back end functions as an interpreter rather than a complier, and therefore the back end has some limitations associated with interpreted languages. For example, user input is not possible in the generated HTML code because this input is encountered after the interpretation. Also, semantic errors are only detected by the back end if the errors are actually executed by the back end ("dead" code is not checked for semantic errors).

For 2.c, HTML functions, the tree walker dispatches calls to the HGLCommon class (see) which determines which HTML object the function was called on, extracts its arguments, and calls the function, using Java reflection techniques, on the appropriate Java HTML class (see Figure 17).

The Java HTML framework used by HGL was designed such that the back end (tree walker) need not have any knowledge of individual HTML functions implemented by the framework. For example, functions like .set(), .add() or .addElement() are all member functions of some HglHGMLObject class. The functions are prefixed in such a way that they can be found by reflection by HGLCommon code. In this way, new functions can be added without modifying the back end. When a function is called which does not exist, the framework will throw an exception which can be caught and handled by the back end. (See Figure 18). The tree walker was coded by Tim Kaczynski, Yan Koyfman and Ed Mezarina. The HTML class subsystem was written by Yan Koyfman.

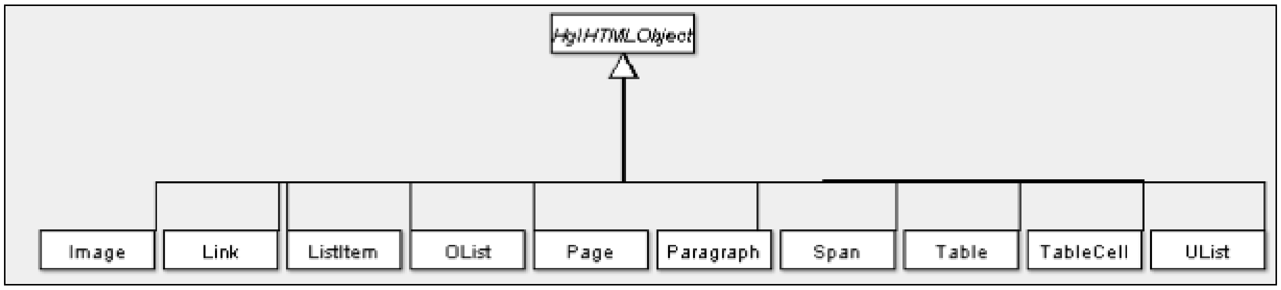


Figure 17: HTML Classes UML Diagram

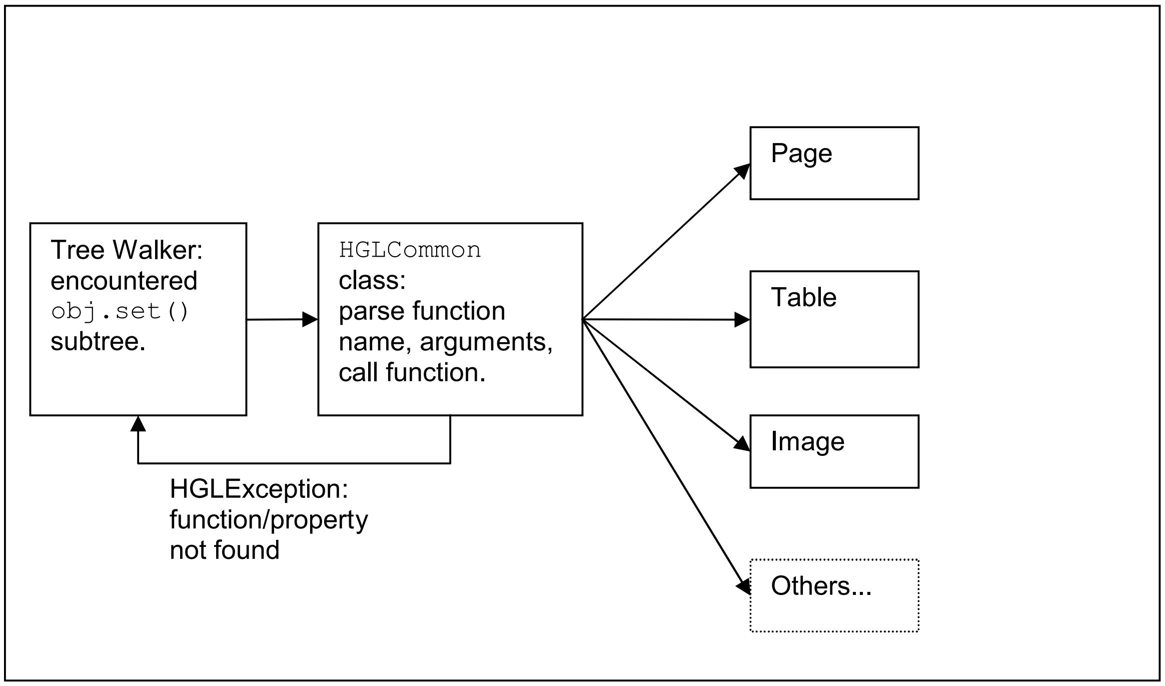

Tree Walker: encountered obj.set() subtree.
HGLCommon class: parse function name, arguments, call function.
HGLException: function/property not found
Page
Table
Image
Others...

Figure 18: HTML Function Dispatch

5.3 Logger and Error Handler

A HGLLogFormatter class appends the raw messages logged with either function name or class name to messages logged to the common Logger object shared by all parts of the translator. During development, we used the –d flag to the translator to print debugging messages to the console. The logger writes all messages to the screen and to two rotated files.

The exception class heirarchy is shown in Figure 19. HGLException exists primarily to distinguish exceptions thrown by HGL code versus those that occur due to JVM errors like null pointer exceptions or class cast exceptions.

The HGLExceptionLine class and its subclasses contain line number information. These exceptions are sent to a function in HGLCommon called `scold()` that prints out an error or warning message. This was done to ensure a consistent error message output format. Error handling in the parser is limited to catching exceptions after each main rule and re-directing it to the common error output routine for printing on the console.

The logger and error handlers were implemented by Shruti Gandhi, Yan Koyfman and Tim Kaczynski.

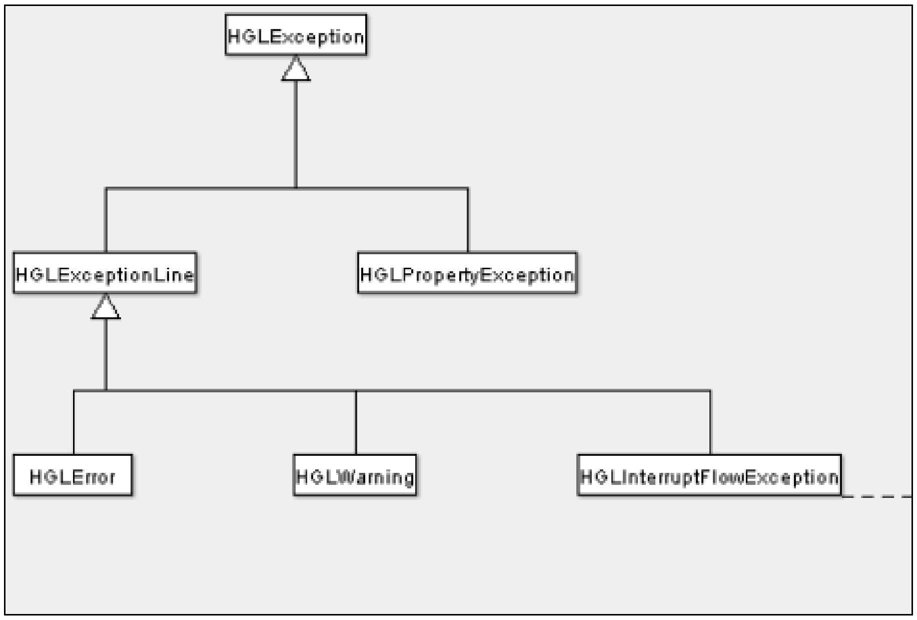


Figure 19: Exception class Hierarchy

6.0 Testing Plan

6.1 Unit Test

The following tests were considered for unit test:

- Comment and white spaces,
- Identifier formation correctness (numerical and string characters),
- Declaration, and dereferencing of variables,
- String declaration assignment,
- Operator evaluation correctness,
- Iterative statements formation and execution,
- Array definitions([]),
- Group block statements ({}),
- Function syntax, and
- Data type validity.

These tests also formed part of the regression bucket described below.

6.2 Function Test

Variations Identification process involved two steps:

1. Testing of each functionality mentioned in the Language Reference Manual
2. Testing the implementation of the functions themselves.

For each of the section above, individual testcases were identified to test each case and the expected resulting output was documented with a reason. As a follow on step, testcases were implemented and its output was matched with the expected output to declare its success or failure. The failures were then identified to fix bugs in our code. We also wrote a regression bucket to rerun our testcases after any changes to the code were made.

The table below lists unique test-cases to verify each HGL function. Each number in the first column corresponds to a unique file (tc*.hgl) that is compared against expected (correct) output. For testcase 1, for example:
Source: tc1.hgl
Result: out1.html
Expected Result: expect1.html

#	Description	Result	Reason
1*	Make sure comments start with //.	Success.	
2	WhiteSpace ignored.	Success.	Trailing white space not ignored. See 1
3*	Test identifier declaration.		

	Create identifiers with mixed case letters. Create identifiers with numbers. Create identifiers with underscore.	
	Create test with following keywords to make sure they are inbuilt keywords.	
4a	“boolean”	Success.
4b*	“true”	Success.
4b*	“false”	Success.
4b*	Break	
4c*	“if” (test to print an int if the 'if' condition evaluates to true and do nothing if it fails.)	Success.
4d*	“else”	Success.
4e*	“int”	Success.
4f*	“string”	
4g*	While condition	Success.
4g*	Ulist	Success.
4g*	Olist	Success.
5	Test to verify if you can override the variable values from outside and inside blocks.	Success.
	Test the validity of the following operators:	
6a*	+	Success.
6b*	-	Success.
6c*	=	Success.
6c*	/	Success.
6c*	*	Success.
6c*	<	Success.
6c*	>	Success.
6c*	<=	Success.
6c*	>=	Success.
6c*	“==”	Success.
6c*	!=	Success.
6c*	||	Success.
6c*	&&	
7*	[] for array definitions , to separate elements in the list	Success

8a*	Test the validity of the following datatypes.	Success.	
	“page”	Success.	
	“paragraph”	Success.	
	“image”	Success.	
	“array”	Success.	
	CSS Property Types	Success.	
	“font”	Success.	
	"font-size"	Success.	
	border	Success.	
	fgcolor	Success.	
	bgcolor	Success.	
	bgimage	Success.	
	border-right	Success.	
	width	Success.	
	margin-left	Success.	
	border-right	Success.	
	line-height	Success.	
	font-family	Success.	
	addr	Success.	
	margin-top	Success.	
	top	Success.	
	vertical-align	Success.	
	. (dot for float and function)	Success.	
	| (String concatenation)	Success.	
8b	- a few invalid types (they don't work)	Success.	
	Function		
9a*	Function calls.	Success.	
9a*	Return	Success.	
9b	Argument validation (type, number)	Success.	Function invocation had no return value.
	Scoping (global vs. local)		
10a	Variable defined twice in same scope.	Success.	value already defined”
10b*	Variable defined in global scope and then in an if block and test if the correct values are used.	Success.	
10c	Variable not accessible outside of the scope	Success.	Undeclared variable
		Success.	

Misc		Proved through other testcases	
	1	Check use of the follows: {} to group block statements () Enclose function arguments, expressions	Success.
	2	$ when followed by a keyword, denote a variable	Success.

6.3 Regression Test

6.3.1 Test bucket

The test bucket consists of a series of test selected from unit test and function test. The tests are modular in nature to allow for better understanding and analysis of test categories such as function analysis, object analysis, and expression analysis, among others. The test case descriptions described in section 6.1 and 6.2 are all part of the regression bucket. The test cases and test automation framework was coded by Shruti Gandhi.

6.3.2 Automation

The regression test suite has been automated by using shell scripts that

- Runs the test
- Compare the output of the test with the expected output.
- Finally, lists all the tests that failed with their resulting errors.

The following are the scripts used:

1. ***run_tests.sh:*** One script to invoke the below scripts.
2. ***run_scripts.sh***: Executes each test scenario and produces out*.html output files that will later be compared to the expected expect*.html outputs.
```
Example:
java hgl -oout10a.html tc10a.hgl > runerror.out 2>&1
java hgl -oout10b.html tc10b.hgl >> runerror.out 2>&1
```

3. ***testscript.sh***: Performs a comparison between the outputs generated by the run_script.sh and the respective expected outcomes. When a test fails you get an error message as below:
```
Example:
diff -w -b -B out1.html expect1.html > /dev/null 2>&1; if [ $? -ne
0 ]; then echo "tc1.hgl failed!"; fi
diff -w -b -B out1.html expect1.html > /dev/null 2>&1; if [ $? -ne
0 ]; then echo "tc1.hgl failed!"; fi
```

Output:
```
'tc6c.hgl failed!'
```

6.3.3 Examples

TestScenario: tc9a.hgl

```
//---------------------------
//tc9a.hgl: Test functions
//          calls.
//---------------------------
int fun(int a, int b, Paragraph p)
{
  while ($a < 5)
  {
   if($a == 3)
    {
      break;

    }

    $a = $a + 1;
  }

  $p.add($a);
  $p.add($b);

  return $a;

}

Paragraph parb;
int retval;
$retval = fun(1, 2, $parb);

Page page;
$page.add($parb);
$page.print();
```

Output: out11.html

```
<html><head>
<style type="text/css" title="hglCSS"
media="all">
</style>

</head>

<body>
  <p>
    3
    2
  </p>

</body>

</html>
```

Expected: expected11.html

```
<html><head>
<style type="text/css" title="hglCSS"
media="all">
</style>

</head>

<body>
  <p>
    3
    2
  </p>

</body>

</html>
```

Diff results = 0 (exact match / test case status: SUCCESS)

TestScenario: tc10b.hgl

```
//----------------------------
//tc10b.hgl: Test scope
//
//----------------------------
int value;
$value = 0;
Page p;

if($value == 0){
   int value;
   $value = 2;
   $p.add("Printing the new local int value from within the if block: ");
   $p.add($value);
   $p.add("<br>");

}

int i;
$i=0;
while($i == 0){
   int value;
   $value = 2;
   $p.add("Printing the new local int value from within the while block:
");
   $p.add($value);
   $p.add("<br>");
   $i = $i +1;

}
```

```
$i=0;
while($i == 0){

   $p.add("Printing the outer int value from within the while block: ");
   $p.add($value);
   $p.add("<br>");
   $i = $i +1;

}

//test for boolean value false
$p.add("Printing int value from the outer scope: ");
$p.add($value);
$p.print();
```

Output: out12.html

```
<html><head>
<style type="text/css" title="hglCSS"
media="screen">
</style>

</head>

<page>
  Printing the new local int value from
within the if block:
  2
  <br>
  Printing the new local int value from
within the while block:
  2
  <br>
  Printing the outer int value from within
the while block:
  0
  <br>
  Printing int value from the outer scope:
  0

</page>

</html>
```

Expected: expected12.html

```
<html><head>
<style type="text/css" title="hglCSS"
media="screen">
</style>

</head>

<page>
  Printing the new local int value from
within the if block:
  2
  <br>
  Printing the new local int value from
within the while block:
  2
  <br>
  Printing the outer int value from within
the while block:
  0
  <br>
  Printing int value from the outer scope:
  0

</page>

</html>
```

Diff results = 0 (exact match / test case status: SUCCESS)

7.0 Lessons Learned

7.1 Shruti Gandhi

Picking a simple project is crucial to better understand the concepts in the class. We spend a few meetings initially to pick an innovative project. It is important to specify the project requirements before you start the project. This helps in defining a clear 'team goal'.

Pair Programming is a good model for development and keeps all team members involved.

7.2 Yan Koyfman

Even in a relatively smoothly running project such as this, some lessons emerged, both reinforcing decisions made at the start and pointing to some things that could have been done better.

On the positive side, the decision to employ project management software to track tasks, in our case the Web based program dotProject, proved useful in its job as a kind of multi-user PDA, reminding us of what remained to be done.

On the revision management front, though we well knew that a source code version control system is essential for a project of this size, we did not fully understand our requirements, and failed to examine alternatives to CVS. Though for most tasks CVS worked very well, it was probably not the best choice due to its inefficient handling of binary files. A newer system, like Subversion, would probably have been better suited to dealing with multiple versions of documents in MS Word and PDF format. (Related aside: Using MS Word to create a collaborative document like this report is tricky. I would advise future groups to consider using LaTeX or other plain-text mark-up based fomat.) To browse the CVS repository graphically, ViewCVS, the web-based CVS browser, proved indespensible.

Apart from the development tools used, I would advise other groups to be sure that all team members have sufficient "buy-in" and understanding of the goal of the project. There was somewhat of an uneven level of expertise in HTML/CSS skills in this group; progress may have accrued even faster if we had taken more time during initial meetings to get everyone more comfortable with the non-syntax/semantics related aspects of the language back end, in this case HTML generation.

7.3 Tim Kaczynski

- Define the scope of your project early and in as much detail as possible. It's easy to get carried away when there is no defined end point, and the basic function often gets neglected.

- ANTLR is your friend... understand the code that it generates. Sometimes it's easier to do tree parsing yourself and if you can work with ANTLR you'll have much less code to write.

- Unit test your code before you integrate it, even if you have decided on having a dedicated function test person. Finding bugs with concurrent development is hard enough as it is.

- Spend some time learning how to re-generate the trees that ANTLR generates... if you know what the tree looks like (and that it looks how you want it to look), the tree walking is much easier.

- The only person who might have a more frustrating job than the project leader is the function tester—but it's your job to challenge both of them when you think something isn't right or could be done better. When done professionally, you'll all remain friends.

7.4 Edward Mezarina

The most important lesson I learned is that establishing clear design/architectural concepts, goals, and procedures among all team members is crucial to maintain focus and schedules. Last, one aspect of the development process that worked for us is the involvement of all team members ("Pair" Programming model) in the development of HGL's front end and certain aspects of the back end.

8.0 Bibliography

[1] dotProject, http://www.dotproject.net/
[2] CSS Reference, http://htmlhelp.com/reference/css/
[3] CSS Reference, http://www.w3.org/TR/REC-CSS2/propidx.html

9.0 Appendix A: CSS Properties supported by HGL [2],[3]

9.1.1 Font

Syntax:	font: <value>
Possible Values:	[<font-style> || <font-variant> || <font-weight>]? <font-size> [/ <line-height>]? <font-family>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	Yes
Use - Examples	The **font** property may be used as a shorthand for the various font properties, as well as the line height. For example, `P { font: italic bold 12pt/14pt Times, serif }` specifies paragraphs with a bold and italic Times or serif font with a size of 12 points and a line height of 14 points.

9.1.2 Font Style

Syntax:	font-style: <value>
Possible Values:	normal | *italic* | *oblique*
Initial Value:	normal
Applies to:	All elements
Inherited:	Yes

	The **font-style** property defines that the font be displayed in one of three ways: **normal**, ***italic*** or ***oblique*** (slanted). A sample style sheet with **font-style** declarations might look like this:
	`H1 { font-style: oblique }`
	`P  { font-style: normal }`

9.1.3 Font Size

Syntax:	font-size: <absolute-size> | <relative-size> | <length> | <percentage>
Possible Values:	• <absolute-size> ○ xx-small | x-small | small | medium | large | x-large | xx-large • <relative-size> ○ larger | smaller • <length> • <percentage> (in relation to parent element)
Initial Value:	medium
Applies to:	All elements
Inherited:	Yes
	The **font-size** property is used to modify the size of the displayed font. Absolute lengths (using units like **pt** and **in**) should be used sparingly due to their weakness in adapting to different browsing environments. Fonts with absolute lengths can very easily be too small or too large for a user. `H1 { font-size: large } P { font-size: 12pt }`

9.1.4 Font Family

Syntax:	font-family: [[<family-name> | <generic-family>],]* [<family-name> | <generic-family>]
Possible Values:	<family-name> • Any font family name may be used <generic-family> • **serif** (*e.g.*, Times) • **sans-serif** (*e.g.*, Arial or Helvetica) • **cursive** (*e.g.*, Zapf-Chancery) • **fantasy** (*e.g.*, Western) • **monospace** (*e.g.*, Courier)
Initial Value:	Determined by browser
Applies to:	All elements
Inherited:	Yes
Use - Examples	Font families may be assigned by a specific font name or a generic font family. Obviously, defining a specific font will not be as likely to match as a generic font family. Multiple family assignments can be made, and if a specific font assignment is made it should be followed by a generic family name in case the first choice is not present. A sample **font-family** declaration might look like this: `P { font-family: "New Century Schoolbook", Times, serif }` Notice that the first two assignments are specific type faces: New Century Schoolbook and Times. However, since both of them are **serif** fonts, the generic font family is listed as a backup in case the system does not have either of these but has another **serif** font which meets the qualifications. Any font name containing whitespace must be quoted, with either single or double quotes. The font family may also be given with the **font** property.

9.1.5 Border

Syntax:	border: <value>
Possible Values:	<border-width> || <border-style> || <color>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No

Use – Examples	The **border** property is a shorthand for setting the width, style, and color of an element's border.
	
	Examples of border declarations include:
	
	H2 { border: groove 3em }
	A:link { border: solid blue }
	A:visited { border: thin dotted #800080 }
	A:active { border: thick double red }
	
	The **border** property can only set all four borders; only one border width and border style may be given. To give different values to an element's four borders, an author must use one or more of the **border-top**, **border-right**, **border-bottom**, **border-left**, **border-color**, **border-width**, **border-style**, **border-top-width**, **border-right-width**, **border-bottom-width**, or **border-left-width** properties.

9.1.6 Left Border

Syntax:	border-left: <value>
Possible Values:	<border-left-width> || <border-style> || <color>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No
Use -Examples	The **border-left** property is a shorthand for setting the width, style, and color of an element's left border. Note that only one **border-style** value may be given. One may also use the **border** shorthand property.

9.1.7 Bottom Border

Syntax:	border-bottom: <value>
Possible Values:	<border-bottom-width> || <border-style> || <color>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No
Use - Examples	The **border-bottom** property is a shorthand for setting the width, style, and color of an element's bottom border. Note that only one **border-style** value may be given. One may also use the **border** shorthand property.

9.1.8 Right Border

Syntax:	border-right: <value>
Possible Values:	<border-right-width> || <border-style> || <color>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No
Use -Examples	The **border-right** property is a shorthand for setting the width, style, and color of an element's right border. Note that only one **border-style** value may be given. One may also use the **border** shorthand property.

9.1.9 Top Border

Syntax:	border-top: <value>
Possible Values:	<border-top-width> || <border-style> || <color>
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No
Use -Examples	The **border-top** property is a shorthand for setting the width, style, and color of an element's top border. Note that only one **border-style** value may be given. One may also use the **border** shorthand property.

9.1.10 Width

Syntax:	width: <value>
Possible Values:	<length> | <percentage> | auto
Initial Value:	auto
Applies to:	Block-level and replaced elements
Inherited:	No
Use - Examples	Each block-level or replaced element can be given a width, specified as a length, a percentage, or as **auto**. (A **replaced element** is one for which only the intrinsic dimensions are known; HTML replaced elements include **IMG**, **INPUT**, **TEXTAREA**, **SELECT**, and **OBJECT**.) The initial value for the **width** property is **auto**, which results in the element's intrinsic width (*i.e.*, the width of the element itself, for example the width of an image). Percentages refer to the parent element's width. Negative values are not allowed. This property could be used to give common widths to some **INPUT** elements, such as submit and reset buttons: `INPUT.button { width: 10em }`

9.1.11 Height

Syntax:	height: <value>
Possible Values:	<length> | auto
Initial Value:	auto
Applies to:	Block-level and replaced elements
Inherited:	No
Use - Examples	Each block-level or replaced element can be given a height, specified as a length or as **auto**. (A **replaced element** is one for which only the intrinsic dimensions are known; HTML replaced elements include **IMG**, **INPUT**, **TEXTAREA**, **SELECT**, and **OBJECT**.) The initial value for the **height** property is **auto**, which results in the element's intrinsic height (*i.e.*, the height of the element itself, for example the height of an image). Negative lengths are not allowed. As with the **width** property, **height** can be used to scale an image: `IMG.foo { width: 40px; height: 40px }` In most cases, authors are advised to scale the image in an image editing program, since browsers will not likely scale images with high quality, and since scaling down causes the user to download an unnecessarily large file. However, scaling through the **width** and **height** properties is a useful option for user-defined style sheets in order to overcome vision problems.

9.1.12 Float

Syntax:	float: <value>
Possible Values:	left | right | none
Initial Value:	none
Applies to:	All elements
Inherited:	No
Use - Examples	The **float** property allows authors to wrap text around an element. This is identical in purpose to HTML 3.2's **ALIGN=left** and **ALIGN=right** for the **IMG** element, but CSS1 allows all elements to "float," not just the images and tables that HTML 3.2 allows.

9.1.13 Top Margin

Syntax:	margin-top: <value>
Possible Values:	<length> | <percentage> | auto
Initial Value:	
Applies to:	All elements
Inherited:	No
Use - Examples	The **margin-top** property sets the top margin of an element by specifying a length or a percentage. Percentage values refer to the parent element's width. Negative margins are permitted. For example, the following rule would eliminate the top margin of a document: `BODY { margin-top: 0 }` Note that adjoining vertical margins are collapsed to use the maximum of the margin values.

9.1.14 Right Margin

Syntax:	margin-right: <value>
Possible Values:	<length> | <percentage> | auto
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **margin-right** property sets the right margin of an element by specifying a length or a percentage. Percentage values refer to the parent element's width. Negative margins are permitted. Example: `P.narrow { margin-right: 50% }` Note that adjoining horizontal margins are not collapsed.

9.1.15 Bottom Margin

Syntax:	margin-bottom: <value>
Possible Values:	<length> | <percentage> | auto
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **margin-bottom** property sets the bottom margin of an element by specifying a length or a percentage. Percentage values refer to the parent element's width. Negative margins are permitted. Example: `DT { margin-bottom: 3em }` Note that adjoining vertical margins are collapsed to use the maximum of the margin values.

9.1.16 Left Margin

Syntax:	margin-left: <value>
Possible Values:	<length> | <percentage> | auto
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use – Examples	The **margin-left** property sets the left margin of an element by specifying a length or a percentage. Percentage values refer to the parent element's width. Negative margins are permitted. Example: `ADDRESS { margin-left: 50% }` Note that adjoining horizontal margins are not collapsed.

9.1.17 Margin

Syntax:	margin: <value>
Possible Values:	[<length> | <percentage> | auto]{1,4}
Initial Value:	Not defined
Applies to:	All elements
Inherited:	No

Use - Examples	The **margin** property sets the margins of an element by specifying between one and four values, where each value is a length, a percentage, or **auto**. Percentage values refer to the parent element's width. Negative margins are permitted.
	
	If four values are given, they apply to top, right, bottom, and left margin, respectively. If one value is given, it applies to all sides. If two or three values are given, the missing values are taken from the opposite side.
	
	Examples of margin declarations include:
	
	`BODY { margin: 5em }             /* all margins 5em */`
	`P    { margin: 2em 4em }         /* top and bottom margins 2em,`
	`                                    left and right margins 4em */`
	`DIV  { margin: 1em 2em 3em 4em } /* top margin 1em,`
	`                                    right margin 2em,`
	`                                    bottom margin 3em,`
	`                                    left margin 4em */`

9.1.18 Top Padding

Syntax:	padding-top: <value>
Possible Values:	<length> | <percentage>
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **padding-top** property describes how much space to put between the top border and the content of the selector. The value is either a length or a percentage. Percentage values refer to the parent element's width. Negative values are *not* permitted.

9.1.19 Right Padding

Syntax:	padding-right: <value>
Possible Values:	<length> | <percentage>
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **padding-right** property describes how much space to put between the right border and the content of the selector. The value is either a length or a percentage. Percentage values refer to the parent element's width. Negative values are *not* permitted.

9.1.20 Bottom Padding

Syntax:	padding-bottom: <value>
Possible Values:	<length> | <percentage>
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **padding-bottom** property describes how much space to put between the bottom border and the content of the selector. The value is either a length or a percentage. Percentage values refer to the parent element's width. Negative values are *not* permitted.

9.1.21 Left Padding

Syntax:	padding-left: <value>
Possible Values:	<length> | <percentage>
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **padding-left** property describes how much space to put between the left border and the content of the selector. The value is either a length or a percentage. Percentage values refer to the parent element's width. Negative vals not permitted.

9.1.22 Padding

Syntax:	padding: <value>
Possible Values:	[<length> | <percentage>]{1,4}
Initial Value:	0
Applies to:	All elements
Inherited:	No
Use - Examples	The **padding** property is a shorthand for the **padding-top**, **padding-right**, **padding-bottom**, and **padding-left** properties. An element's **padding** is the amount of space between the border and the content of the element. Between one and four values are given, where each value is either a length or a percentage. Percentage values refer to the parent element's width. Negative values are *not* permitted. If four values are given, they apply to top, right, bottom, and left padding, respectively. If one value is given, it applies to all sides. If two or three values are given, the missing values are taken from the opposite side. For example, the following rule sets the top padding to 2em, the right padding to 4em, the bottom padding to 5em, and the left padding to 4em: `BLOCKQUOTE { padding: 2em 4em 5em }`

9.1.23 Top

Syntax:	top: <value>
Possible Values:	<length> | <percentage> | auto | inherit
Initial Value:	auto
Applies to:	Positioned elements
Inherited:	No
Use - Examples	The **top** property specifies how far a box's top content edge is offset below the top edge of the box's containing block. **<length>** The offset is a fixed distance from the reference edge. **<percentage>** The offset is a percentage of the containing block's width (for 'left' or 'right') or height (for 'top' and 'bottom'). For 'top' and 'bottom', if the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like 'auto'. **auto** The effect of this value depends on which of related properties have the value 'auto' as well. See the sections on the width and height of absolutely positioned, non-replaced elements for details.

9.1.24 Right

Syntax:	right: <value>
Possible Values:	<length> | <percentage> | auto | inherit
Initial Value:	Auto
Applies to:	Positioned ements
Inherited:	No
Use - Examples	The **right** property specifies how far a box's right content edge is offset to the left of the right edge of the box's containing block. **<length>** The offset is a fixed distance from the reference edge. **<percentage>** The offset is a percentage of the containing block's width (for 'left' or 'right') or height (for 'top' and 'bottom'). For 'top' and 'bottom', if the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like 'auto'. **auto** The effect of this value depends on which of related properties have the value 'auto' as well. See the sections on the width and height of absolutely positioned, non-replaced elements for details.

9.1.25 Bottom

Syntax:	**bottom: <value>**
Possible Values:	**<length> | <percentage> | auto | inherit**
Initial Value:	**auto**
Applies to:	**Positioned elements**
Inherited:	**No**
Use - Examples	**The bottom specifies how far a box's bottom content edge is offset above the bottom of the box's containing block.** **<length>** **The offset is a fixed distance from the reference edge.** **<percentage>** **The offset is a percentage of the containing block's width (for 'left' or 'right') or height (for 'top' and 'bottom'). For 'top' and 'bottom', if the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like 'auto'.** **auto** **The effect of this value depends on which of related properties have the value 'auto' as well. See the sections on the width and height of absolutely positioned, non-replaced elements for details.**

9.1.26 Left

Syntax:	left: <value>
Possible Values:	<length> | <percentage> | auto | inherit
Initial Value:	auto
Applies to:	Positioned elements
Inherited:	No
Use – Examples	The **left** specifies how far a box's left content edge is offset to the right of the left edge of the box's containing block. **<length>** The offset is a fixed distance from the reference edge. **<percentage>** The offset is a percentage of the containing block's width (for 'left' or 'right') or height (for 'top' and 'bottom'). For 'top' and 'bottom', if the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like 'auto'. **auto** The effect of this value depends on which of related properties have the value 'auto' as well. See the sections on the width and height of absolutely positioned, non-replaced elements for details.

9.1.27 Background

Syntax:	background: <value>
Possible Values:	[<'background-color'> || <'background-image'> || <'background-repeat'> || <'background-attachment'> || <'background-position'>] | inherit
Initial Value:	not defined for shorthand properties
Applies to:	All elements
Inherited:	No
Use - Examples	The 'background' property is a shorthand property for setting the individual background properties (i.e., 'background-color', 'background-image', 'background-repeat', 'background-attachment' and 'background-position') at the same place in the style sheet. The 'background' property first sets all the individual background properties to their initial values, then assigns explicit values given in the declaration. In the first rule of the following example, only a value for 'background-color' has been given and the other individual properties are set to their initial value. In the second rule, all individual properties have been specified. BODY { background: red } P { background: url("chess.png") gray 50% repeat fixed }

9.1.28 Line Height

Syntax:	line-height: <value>
Possible Values:	normal | <number> | <length> | <percentage>
Initial Value:	normal
Applies to:	All elements
Inherited:	Yes
Use - Examples	The **line-height** property will accept a value to control the spacing between baselines of text. When the value is a number, the line height is calculated by multiplying the element's font size by the number. Percentage values are relative to the element's font size. Negative values are not permitted. Line height may also be given in the **font** property along with a font size. The **line-height** property could be used to double space text: `P { line-height: 200% }` Microsoft Internet Explorer 3.x incorrectly treats number values and values with **em** or **ex** units as pixel values. This bug can easily make pages unreadable, and so authors should avoid provoking it wherever possible; percentage units are often a good choice.

9.1.29 Color (hgl: fgcolor)

Syntax:	color: <color>
Initial Value:	Determined by browser
Applies to:	All elements
Inherited:	Yes

Use-Examples	The **color** property allows authors to specify the color of an element. See the Units section for color value descriptions. Some example color rules include:
	
	`H1 { color: blue }`
	`H2 { color: #000080 }`
	`H3 { color: #0c0 }`
	
	To help avoid conflicts with user style sheets, **background** and **color** properties should always be specified together.

9.1.30 Background Color (HGL: bgcolor)

Syntax:	background-color: <value>
Possible Values:	<color> | transparent
Initial Value:	transparent
Applies to:	All elements
Inherited:	No
Use – Examples	The **background-color** property sets the background color of an element. For example: BODY { background-color: white } H1 { background-color: #000080 } To help avoid conflicts with user style sheets, **background-image** should be specified whenever **background-color** is used. In most cases, **background-image: none** is suitable. Authors may also use the shorthand **background** property, which is currently better supported than the **background-color** property.

9.1.31 Background Image

Syntax:	background-image: <value>
Possible Values:	<url> | none
Initial Value:	none
Applies to:	All elements
Inherited:	No
Use – Examples	The **background-image** property sets the background image of an element. For example: BODY { background-image: url(/images/foo.gif) } P { background-image: url(http://www.htmlhelp.com/bg.png) } When a background image is defined, a similar background color should also be defined for those not loading images. Authors may also use the shorthand **background** property, which is currently better supported than the **background-image** property.

9.1.32 Vertical Alignment

Syntax:	vertical-align: <value>
Possible Values:	baseline | $_{\text{sub}}$ | $^{\text{super}}$ | top | text-top | middle | bottom | text-bottom | <percentage>
Initial Value:	baseline
Applies to:	Inline elements
Inherited:	No

Use - Examples	The **vertical-align** property may be used to alter the vertical positioning of an inline element, relative to its parent element or to the element's line. (An **inline element** is one which has no line break before and after it, for example, **EM**, **A**, and **IMG** in HTML.)
	
	The value may also be a keyword. The following keywords affect the positioning relative to the parent element:
	
	• **baseline** (align baselines of element and parent)
	• **middle** (align vertical midpoint of element with baseline plus half the x-height--the height of the letter "x"--of the parent)
	• **sub** (subscript)
	• **super** (superscript)
	• **text-top** (align tops of element and parent's font)
	• **text-bottom** (align bottoms of element and parent's font)
	
	The keywords affecting the positioning relative to the element's line are
	
	• **top** (align top of element with tallest element on the line)
	• **bottom** (align bottom of element with lowest element on the line)
	
	The **vertical-align** property is particularly useful for aligning images. Some examples follow:
	
	`IMG.middle { vertical-align: middle }`
	`IMG         { vertical-align: 50% }`
	`.exponent   { vertical-align: super }`

9.1.33 Position

Syntax:	position <value>
Possible Values:	static | relative | absolute | fixed | inherit
Initial Value:	static
Applies to:	all elements, but not to generated content
Inherited:	No

Use – Examples	The **position** property applies to all elements.
	
	Some examples follow:
	
	`H1 { position: absolute }`
	
	`#outer {`
	`  position: absolute;`
	`}`

9.1.34 Z-index

Syntax:	position <value>
Possible Values:	auto | <integer> | inherit
Initial Value:	Automatic
Applies to:	Positioned elements
Inherited:	No

Use – Examples	For a positioned box, the 'z-index' property specifies:
	
	1. The stack level of the box in the current stacking context.
	2. Whether the box establishes a local stacking context.
	
	`<IMG id="image" class="pile"`
	`     src="butterfly.gif" alt="A butterfly image"`
	`     style="z-index: 1">`

9.1.35 Text Alignment

Syntax:	text-align: <value>
Possible Values:	left | right | center | justify
Initial Value:	Determined by browser
Applies to:	Block-level elements
Inherited:	Yes

Use – Examples	The **text-align** property can be applied to block-level elements (**P**, **H1**, etc.) to give the alignment of the element's text. This property is similar in function to HTML's **ALIGN** attribute on paragraphs, headings, and divisions.
	
	Some examples follow:
	
	`H1          { text-align: center }`
	`P.newspaper { text-align: justify }`

10.0 Appendix B: Code Listings

Source code listings for examples and the HGL translator code.

10.1 Examples

10.1.1 pbc.hgl

Source code for web site depicted in Figure 11

Table 1

```
// Make HTML for front page of pokbooks.com

// Main page
Page index;
$index.position = "relative";

$index.setTitle("Poughkeepsie Book Club");

// Top margin for all boxes
int topMargin;
$topMargin = 10;

// Height (in pixels) of main page constructs
int mainHeight;
$mainHeight = 500;

// Left margin (in pixels) of main box
int mainLeft;
$mainLeft = 50;

// Width(in pixels) of main box
int mainWidth;
$mainWidth = 900;

// An outer container for parts of the page
Paragraph box;
$box.border = "thin solid black";
$box.bgcolor = "transparent";
$box.position = "absolute";
$box.left = $mainLeft | "px";
$box.width = $mainWidth | "px";
$box.margin-top = $topMargin | "px";
$box.height = $mainHeight | "px";

// Left navigation pane
Paragraph navPane;
int navWidth;
$navWidth=175;

$navPane.bgcolor = "#ffffcc";
```

```
$navPane.border = "thin solid gray";
$navPane.position = "absolute";
$navPane.left = $mainLeft | "px";
$navPane.width = $navWidth | "px";
$navPane.margin-top = $topMargin | "px";
$navPane.height = $mainHeight | "px";

// Font sizes for title
int fontLarge;
int fontMedium;
int pad;

$fontLarge = 25;
$fontMedium = 12;
$pad = 5;

//////////////////////////////////////////////////////////////////////
// Title section
//////////////////////////////////////////////////////////////////////

Paragraph titlePane;
$titlePane.bgcolor = "#F0E9AA";
$titlePane.font = "normal normal bolder " | $fontLarge | "px Photina Casual
Black";
$titlePane.margin-top = $topMargin | "px";
$titlePane.border = "thin solid gray";
$titlePane.position = "absolute";
$titlePane.left = 1+$navWidth+$mainLeft | "px";
$titlePane.width = $mainWidth-($navWidth) | "px";

// Title pane text objects
Paragraph title_1;
$title_1.inherit($titlePane);
$title_1.fgcolor = "#0E6CAF";
$title_1.add("Poughkeepsie Book Club");

Paragraph title_2;
$title_2.inherit($title_1);
$title_2.margin-top = $topMargin+$fontLarge+$pad | "px";
$title_2.font = "normal normal bolder " | $fontMedium | "px Photina Casual
Black";
$title_2.border = "none";
$title_2.border-left = "thin solid gray";
$title_2.border-right = "thin solid gray";
$title_2.border-bottom = "thin solid gray";
$title_2.add("Come for the pizza, stay for the pants!");

Paragraph mtgDate;
$mtgDate.inherit($title_1);
$mtgDate.set("margin-top",
        $topMargin+$fontLarge+$pad+$fontMedium+$pad | "px");
$mtgDate.font-size = $fontLarge | "px";
$mtgDate.add("Next Meeting: Wednesday January 25, 2006 at 7:00PM");

// Add title text to title box
$titlePane.add($title_1);
$titlePane.add($title_2);
```

```
$titlePane.add($mtgDate);

////////////////////////////////////////////////////////////////////////
// Create links
///////////////////////////////////////////////////////////////////////

int idx;
Link navLinks[6];

// Set up format for links
$idx = 0;
$navLinks[$idx].margin-left = "10%";
$navLinks[$idx].line-height = "1.1";
$navLinks[$idx].background = "transparent";
$navLinks[$idx].fgcolor = "green";
$navLinks[$idx].vertical-align = "top";
$navLinks[$idx].font-size = "110%";
$navLinks[$idx].font-family = """Trebuchet MS"", ""Avant Garde"", sans-
serif";
$navLinks[$idx].addr = "phpBB2/index.php";
$navLinks[$idx].add("Forums");
$navLinks[$idx].setPseudo("hover","fgcolor","black");
$navLinks[$idx].setPseudo("hover","border","thin solid gray");
$navLinks[$idx].setPseudo("focus","fgcolor","red");
$navLinks[$idx].setPseudo("active","fgcolor","green");

$idx = $idx+1;
$navLinks[$idx].addr = "phpBB2/rss.php";
$navLinks[$idx].add("(RSS)");

$idx = $idx+1;
$navLinks[$idx].addr = "faq.html";
$navLinks[$idx].add("Info & FAQs");

$idx = $idx+1;
$navLinks[$idx].addr = "books.html";
$navLinks[$idx].add("Our Books");

$idx = $idx+1;
$navLinks[$idx].addr = "gallery.html";
$navLinks[$idx].add("Members");

$idx = $idx+1;
$navLinks[$idx].addr = "about.html";
$navLinks[$idx].add("History");

$idx=1;
while ( $idx < arraySize($navLinks) ) {
  $navLinks[$idx].link($navLinks[0]);
  $idx=$idx+1;
} // while ( $idx < (arraySize($navLinks)-1) )

///////////////////////////////////////////////////////////////////////
// Create containers for current and next book
///////////////////////////////////////////////////////////////////////

Paragraph currBookBox;
Paragraph nextBookBox;
int bkWidth;
int bkTopMargin;
int bkHeight;

$bkWidth = ($mainWidth - $navWidth) / 2;
$bkTopMargin = $topMargin+$fontLarge+$pad+$fontMedium+$pad+$fontLarge;
$bkHeight = $mainHeight - ($bkTopMargin - $topMargin);

////////////////////////////////////////////////////////////////////////
// Current book's container
////////////////////////////////////////////////////////////////////////

$currBookBox.inherit($box);
$currBookBox.left = $mainLeft + $navWidth | "px";
$currBookBox.width = $bkWidth | "px";
$currBookBox.margin-top = $bkTopMargin | "px";
$currBookBox.height = $bkHeight | "px";
$currBookBox.bgcolor = "white";

////////////////////////////////////////////////////////////////////////
// Current book text
////////////////////////////////////////////////////////////////////////

// Month
Paragraph currBook;
$currBook.inherit($currBookBox);
$currBook.font-size = $fontMedium|"px";
$currBook.bgcolor = "#ffffcc";
$currBook.fgcolor = "black";
$currBook.margin-top = $bkTopMargin+$pad | "px";
$currBook.height = $fontMedium+$pad | "px";
$currBook.add("Decanuary's Book");

$currBookBox.add($currBook);

// Title
Paragraph currBookTitle;
$currBookTitle.inherit($currBook);
$currBookTitle.margin-top = $bkTopMargin+$pad+$fontMedium+$pad | "px";
$currBookTitle.font-size = $fontMedium*2|"px";
$currBookTitle.height = $fontMedium*2+$pad | "px";
$currBookTitle.fgcolor = "#02008A";
$currBookTitle.bgcolor = "white";
$currBookTitle.border = "0 none transparent";
$currBookTitle.border-left = "thin solid black";
$currBookTitle.border-right = "thin solid black";
$currBookTitle.add("American Gods");

$currBookBox.add($currBookTitle);

// Author
Paragraph currBookAuthor;
$currBookAuthor.inherit($currBook);
$currBookAuthor.margin-top = $bkTopMargin+$fontMedium*3+$pad*3 | "px";
$currBookAuthor.font-size = $fontMedium|"px";
$currBookAuthor.height = $fontMedium+$pad | "px";
```

```
$currBookAuthor.border-right = "thin solid black";
$currBookAuthor.bgcolor = "white";
$currBookAuthor.border = "0 none transparent";
$currBookAuthor.add("by Neil Gaimon");

$currBookBox.add($currBookAuthor);


//////////////////////////////////////////////////////////////////////
// Current book image
//////////////////////////////////////////////////////////////////////

Image currBookImg;
$currBookImg.top = $bkTopMargin+$fontMedium+$pad+($bkHeight/4) | "px";
$currBookImg.left = $mainLeft + $navWidth + ($bkWidth/2) | "px";
$currBookImg.src = "books/book1205.jpg";
$currBookImg.position = "absolute";

$currBookBox.add($currBookImg);

//////////////////////////////////////////////////////////////////////
// Current book amazon links
//////////////////////////////////////////////////////////////////////

Paragraph currBookAmazon;
$currBookAmazon.inherit($currBookImg);
$currBookAmazon.left = $mainLeft + $navWidth + $pad | "px";
$currBookAmazon.add("<iframe
src=""http://rcm.amazon.com/e/cm?t=poughkeepsibo-
20&o=1&p=8&l=as1&asins=0380789035&fc1=000000&=1&lc1=0000ff&bc1=000000&lt1=_
blank&IS2=1&bg1=ffffff&f=ifr"" style=""width:120px;height:240px;""
scrolling=""no"" marginwidth=""0"" marginheight=""0""
frameborder=""0""></iframe>");

$currBookBox.add($currBookAmazon);

//////////////////////////////////////////////////////////////////////
// Current book library link
//////////////////////////////////////////////////////////////////////

Link currBookLibrary;
$currBookLibrary.inherit($currBookAuthor);
$currBookLibrary.top = $mainHeight-$fontMedium | "px";
$currBookLibrary.margin-top = 0 | "px";
$currBookLibrary.vertical-align = "none";
$currBookLibrary.addr =
"http://gigcat.midhudson.org/search/Xamerican+gods&searchscope=1&SORT=D/Xam
erican+gods&searchscope=1&SORT=D/1%2C4%2C4%2CB/frameset&FF=Xamerican+gods&s
earchscope=1&SORT=D&4%2C4%2C";
$currBookLibrary.font-family = """Trebuchet MS"", ""Avant Garde"", sans-
serif";
$currBookLibrary.fgcolor = "green";
$currBookLibrary.line-height = "1.1";
$currBookLibrary.border = "none";
$currBookLibrary.border-right = "0 solid white";
$currBookLibrary.width = "auto";
$currBookLibrary.margin-left = "10px";
```

```
$currBookLibrary.setPseudo("hover","fgcolor","black");
$currBookLibrary.setPseudo("hover","border","thin solid gray");
$currBookLibrary.setPseudo("focus","fgcolor","red");
$currBookLibrary.setPseudo("active","fgcolor","green");

$currBookLibrary.add("Check the Mid-Hudson Library System");

$currBookBox.add($currBookLibrary);

//////////////////////////////////////////////////////////////////////
// Next book's container
//////////////////////////////////////////////////////////////////////

$nextBookBox.inherit($currBookBox);
$nextBookBox.left = $bkWidth + $mainLeft + $navWidth | "px";

//////////////////////////////////////////////////////////////////////
// Next book text
//////////////////////////////////////////////////////////////////////

// Month
Paragraph nextBook;
$nextBook.inherit($currBook);
$nextBook.left = $bkWidth + $mainLeft + $navWidth | "px";
$nextBook.add("February's Book");

$nextBookBox.add($nextBook);

// Title
Paragraph nextBookTitle;
$nextBookTitle.inherit($currBookTitle);
$nextBookTitle.left = $bkWidth + $mainLeft + $navWidth | "px";
$nextBookTitle.border = "0 none transparent";
$nextBookTitle.border-left = "thin solid black";
$nextBookTitle.add("Freakonomics");

$nextBookBox.add($nextBookTitle);

// Author
Paragraph nextBookAuthor;
$nextBookAuthor.inherit($currBookAuthor);
$nextBookAuthor.left = $bkWidth + $mainLeft + $navWidth | "px";
$nextBookAuthor.border = "0 none transparent";
$nextBookAuthor.border-left = "thin solid black";
$nextBookAuthor.add("by Steven D. Levitt and Stephen J. Dubner");

$nextBookBox.add($nextBookAuthor);

//////////////////////////////////////////////////////////////////////
// Next book image
//////////////////////////////////////////////////////////////////////

Image nextBookImg;
$nextBookImg.inherit($currBookImg);
$nextBookImg.left = $mainLeft + $navWidth + ($bkWidth*1.5) | "px";
$nextBookImg.src = "books/book0206.jpg";
```

```
$nextBookBox.add($nextBookImg);

$box.add($currBookBox);
$box.add($nextBookBox);

//////////////////////////////////////////////////////////////////////
// Next book amazon links
//////////////////////////////////////////////////////////////////////

Paragraph nextBookAmazon;
$nextBookAmazon.inherit($nextBookImg);
$nextBookAmazon.left = $bkWidth + $mainLeft + $navWidth + $pad | "px";
$nextBookAmazon.add("<iframe
src=""http://rcm.amazon.com/e/cm?t=poughkeepsibo-
20&o=1&p=8&l=as1&asins=006073132X&=1&fc1=000000&IS2=1&lt1=_blank&lc1=0000ff
&bc1=000000&bg1=ffffff&f=ifr"" style=""width:120px;height:240px;""
scrolling=""no"" marginwidth=""0"" marginheight=""0""
frameborder=""0""></iframe>");

$nextBookBox.add($nextBookAmazon);

//////////////////////////////////////////////////////////////////////
// Next book library link
//////////////////////////////////////////////////////////////////////

Link nextBookLibrary;
$nextBookLibrary.inherit($currBookLibrary);
$nextBookLibrary.left = $bkWidth + $mainLeft + $navWidth | "px";
$nextBookLibrary.addr =
"http://gigcat.midhudson.org/search/Xfreakonomics&searchscope=1&SORT=D/Xfre
akonomics&searchscope=1&SORT=D&extended=0/1%2C2%2C2%2CB/frameset&FF=Xfreako
nomics&searchscope=1&SORT=D&2%2C2%2C";
$nextBookLibrary.border-right = "0 solid white";
$nextBookLibrary.add("Check the Mid-Hudson Library System");

$nextBookBox.add($nextBookLibrary);

// Add graphic to navigation page
Image mojo;
$mojo.src = "mojo/mojo9.gif";
$mojo.position = "absolute";
$mojo.left = $mainLeft | "px";
$mojo.margin-top = $topMargin | "px";
$mojo.z-index = "8";

// Add links to navigation pane
//$navPane.add($mojo);
$index.add($mojo);
$navPane.newline(2);

Span Home;
$Home.inherit($navLinks[0]);
$Home.fgcolor = "gray";
$Home.setPseudo("hover","border","0 solid gray");
$Home.add("Home");

// Mojo's quote
```

```
Span quote;
$quote.text-align = "center";
$quote.font-style = "italic";
$quote.margin-left = "10%";
$quote.top = "185px";
$quote.position = "absolute";
$quote.add("""Don't make me smite you, lowly primate!""");

// Add HGL logo
Paragraph HGL;
$HGL.inherit($quote);
$HGL.top = $topMargin*2.5+$mainHeight | "px";
$HGL.font-family = "Verdana";
$HGL.font-size = $topMargin | "px";
$HGL.margin-left = $mainWidth-50 | "px";
$HGL.add("Powered by HGL");

$index.add($HGL);

$navPane.add($quote);
$navPane.newline(11);

$navPane.add($Home);
$navPane.newline(2);

$idx=0;
while ($idx<arraySize($navLinks)) {
  $navPane.add($navLinks[$idx]);
  if ($idx != 0)
   $navPane.newline(2);
  $idx=$idx+1;
} // whle ($idx<arraySize($navLinks))

// Add to outer border
$box.add($navPane);
$box.add($titlePane);

$index.add($box);

// Output page
$index.print();
```

10.2 HGL Translator Code Listing

10.2.1 Parser

```
--------------------
---- ./HGLScanner.g
Author: Tim Kazynski, Yan Koyfman, Edward Mezarina, Shruti Gandhi
This file contains the scanner, parser and tree walker.
--------------------
// HGL Group
// COMS 4115 Fall 2005
// References:
// C BNF Grammar: http://lists.canonical.org/pipermail/kragen-hacks/1999-
October/000201.html
// ANTLR C Grammar: http://www.antlr.org/grammar/cgram/grammars/StdCParser.g
// PHP Language Reference: http://docs.php.net/en/langref.html

/*------------------------------------------------------------------*/
/* Lexer                                                            */
/*------------------------------------------------------------------*/
class HGLLexer extends Lexer;
options
{
    testLiterals = false;
    k = 2;
    charVocabulary = '\3'..'\377';
}

tokens {
    BOOLEAN          = "boolean"         ;
    CHAR             = "char"            ;
    CONST            = "const"           ;
    ELSE             = "else"            ;
    IF               = "if"              ;
    INTEGER          = "int"             ;
    FLOAT            = "float"           ;
    STR              = "string"          ;
    WHILE            = "while"           ;
    HTML_START;                         // between this and HTML_END only HTML types.
    PARAGRAPH        = "Paragraph"       ; // String should map to Java class name
    OLIST            = "OList"           ;
    ULIST            = "UList"           ;
    PAGE             = "Page"            ;
    TABLE            = "Table"           ;
    IMAGE            = "Image"           ;
    LINK             = "Link"            ;
    SPAN             = "Span"            ;
    HTML_END;                           // one plus end of HTML types
    CONTINUE         = "continue"        ;
    BREAK            = "break"           ;
    TRUE             = "true"            ;
    FALSE            = "false"           ;
    NULL             = "null"            ;
    RETURN           = "return"          ;
    EXPR;
    FLOAT_LIT;
    INT_LIT;
    START;
    DECL;
    ARRAY_DECL;
    FUNDECL;
    STMT;
    CSTMT;
    ARRAYINIT;
    ARRAYDEREF;
    FUNCTIONCALL;
    FUNCTIONCALLOBJ;
    FUNCTIONCALLSET;
    FUNDEF;
    PARM_LIST_TYPE;
    PARM_LIST;
    UMINUS;
    UPLUS;
}

PLUS options {paraphrase="+";}           : '+' ;
MINUS options {paraphrase="-";}          : '-' ;
TIMES options {paraphrase="*";}          : '*' ;
DIV options {paraphrase="/";}            : '/' ;
ASSIGN options {paraphrase="=";}         : '=' ;
ARRAY_OPEN options {paraphrase="[";}     : '[' ;
ARRAY_CLOSE options {paraphrase="]";}    : ']' ;
STR_CAT options {paraphrase="|";}        : '|' ;
PAREN_OPEN options {paraphrase="(";}     : '(' ;
PAREN_CLOSE options {paraphrase=")";}    : ')' ;
BRACE_OPEN options {paraphrase="{";}     : '{' ;
BRACE_CLOSE options {paraphrase="}";}    : '}' ;
SEMI options {paraphrase=";";}           : ';' ;
DOT options {paraphrase=".";}            : '.' ;
COMMA options {paraphrase=",";}          : ',' ;
REL_LT options {paraphrase="<";}         : '<' ;
REL_GT options {paraphrase=">";}         : '>' ;
REL_LTE options {paraphrase="<=";}       : "<=";
REL_GTE options {paraphrase=">=";}       : ">=";
REL_EQ options {paraphrase="==";}        : "==";
REL_NEQ options {paraphrase="!=";}       : "!=";
COND_AND options {paraphrase="&&";}      : "&&";
COND_OR options {paraphrase="||";}       : "||";

protected
LETTER : ('a'..'z' | 'A'..'Z') ;

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

protected
HEX_DIGIT : (DIGIT)|('A'..'F')|('a'..'f');

protected
EXPONENT   : ('e' | 'E') ('+' | '-')? DIGIT
           ;

protected
DIGITS     : (DIGIT)+
           ;
ID options { testLiterals = true; paraphrase="identifier";}:
            (LETTER | '_') (LETTER | DIGIT | '_')*
           ;

PROPID : '.' (LETTER | '_') (LETTER | DIGIT | '_' | '-')*
           ;

VARIABLE options {paraphrase="variable";}   : '$' ID
           ;

NUMBER
    : DIGITS
        ( ('.' (DIGIT)+ (EXPONENT)? | EXPONENT) { $setType(FLOAT_LIT); }
        | /* empty */ { $setType(INT_LIT); }
        )
;

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

```
CHAR_LIT   : '\''! ('"' | ~('"')) '\''!
           ;

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

COMMENT    : "//"
             ((~'\n'))* '\n' { newline(); }
             { $setType(Token.SKIP); }
           ;

/*-----------------------------------------------------------------------*/
/* Parser:                                                               */
/*-----------------------------------------------------------------------*/
class HGLParser extends Parser;
options { buildAST = true; k = 2; defaultErrorHandler=false;}

// Start rule: list of declarations and statements
start
    : ( declaration | statement)+ EOF!
        {#start = #([START, "START"], start);}
    ;
exception // for rule
    catch [RecognitionException ex] {
        HGLCommon.scold(ex);
}
declaration : builtinType ID (arrayDecl  SEMI! | functionCommon)
             {#declaration = #([DECL, "DECL"], declaration);}
            ;
exception // for rule
    catch [RecognitionException ex] {
        HGLCommon.scold(ex);
}

arrayDecl
    : (ARRAY_OPEN! expression ARRAY_CLOSE!)
        {#arrayDecl = #([ARRAY_DECL, "ARRAY_DECL"], arrayDecl);}
    | /*nothing*/
    ;


statement
    : (SEMI                      // NULL statment
        | compoundStatement
        | expression SEMI!                // expression
            // Iteration statements
        | WHILE^ PAREN_OPEN! expression PAREN_CLOSE! statement
            // Alternation statements
        | IF^ PAREN_OPEN! expression PAREN_CLOSE! statement
            (options {greedy=true;} : ELSE! statement)?
        | RETURN^ expression SEMI!
        | BREAK SEMI!
        | CONTINUE SEMI!)
        {#statement = #([STMT, "STMT"], statement);}
    ;
exception // for rule
    catch [RecognitionException ex] {
        HGLCommon.scold(ex);
}

compoundStatement
    : BRACE_OPEN! (statement | declaration)* BRACE_CLOSE!
            { #compoundStatement = #([CSTMT, "CSTMT"], compoundStatement);}

    ;


//-----------------------------------------------------------------------
// Functions:
//-----------------------------------------------------------------------
```

```
functionCommon : PAREN_OPEN! (parmListWithType)? PAREN_CLOSE!
        (SEMI! {#functionCommon = #([FUNDECL, "FUNDECL"], functionCommon);}
        | compoundStatement {#functionCommon = #([FUNDEF, "FUNDEF"], functionCommon);})
    ;

functionCall : ID PAREN_OPEN! (parmList)? PAREN_CLOSE!
        {#functionCall = #([FUNCTIONCALL, "FUNCTIONCALL"], functionCall);}
    ;

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

parmListWithType : builtinType ID (COMMA! builtinType ID)*
        { #parmListWithType = #([PARM_LIST_TYPE, "PARM_LIST_TYPE"], parmListWithType);}
    ;

//----------------------------------------------------------------------
// Expression precedence:
// Lowest : =
//          |
//          ||
//          &&
//          == !=
//          < <= > >=
//          + -
// Highest: * /
//----------------------------------------------------------------------
expression     : assignExp
                 {#expression = #(#[EXPR,"EXPR"],#expression);}
               ;

assignExp      : concatExp (ASSIGN^ assignExp)?
               ;

concatExp      : condOrExp (STR_CAT^ condOrExp)*
               ;

condOrExp      : condAndExp (COND_OR^ condAndExp)*
               ;

condAndExp     : relationalExp (COND_AND^ relationalExp)*
               ;

relationalExp  : relationalExps ((REL_EQ^ | REL_NEQ^) relationalExps)*
               ;

relationalExps :   arithASExp ((REL_LT^  |
                                REL_LTE^ |
                                 REL_GT^ |
                                REL_GTE^ ) arithASExp)*
               ;

arithASExp     :  arithMDExp ((PLUS^ | MINUS^) arithMDExp)*
               ;

arithMDExp     : unaryRules ((TIMES^ | DIV^) unaryRules)*
               ;

unaryRules     : ( (MINUS^) | (PLUS^) )? functionCallObj
               ;

functionCallObj : valAssigned (PROPID
/* funcall */ ((PAREN_OPEN! (parmList)? PAREN_CLOSE! {#functionCallObj =
#([FUNCTIONCALLOBJ, "FUNCTIONCALLOBJ"], functionCallObj);})
/* property */ | {#functionCallObj = #([FUNCTIONCALLSET, "FUNCTIONCALLSET"],
functionCallObj);}))?
    ;

valAssigned
    : functionCall
    | INT_LIT
    | FLOAT_LIT
    | CHAR_LIT
    | STRING
    | VARIABLE {HGLCommon.logger.finer("Matched vbl");}
    | TRUE
    | FALSE
    | NULL
    | PAREN_OPEN! expression PAREN_CLOSE!
    | arrayInitializer
    | arrayDereference
    ;

builtinType     : BOOLEAN | CONST | INTEGER | FLOAT | STR | CHAR | PARAGRAPH |
                  OLIST   | ULIST | PAGE | TABLE | IMAGE | LINK | SPAN
    ;

arrayDereference : VARIABLE ARRAY_OPEN! expression ARRAY_CLOSE!
        {#arrayDereference = #(#[ARRAYDEREF,"ARRAYDEREF"],#arrayDereference);}
    ;

arrayInitializer : ARRAY_OPEN! arrayInitList ARRAY_CLOSE!
        {#arrayInitializer = #(#[ARRAYINIT,"ARRAYINIT"],#arrayInitializer);}

    ;

simpleValue : INT_LIT | FLOAT_LIT | CHAR_LIT | STRING | TRUE | FALSE ;

arrayInitList
    : (simpleValue ( COMMA! simpleValue )*)?

    ;


/*-------------------------------------------------------------------*/
/* TreeWalker:                                                       */
/*-------------------------------------------------------------------*/
class HGLTreeWalker extends TreeParser;

/*-------------------------------------------------------------------*/
/* Static things, inner classes, etc.  Initialization should not be  */
/* done here, but in the START rule instead.                         */
/*-------------------------------------------------------------------*/
{
    /*---------------------------------------------------------------*/
    /* The current symbol table.  As nested scopes are encountered,  */
    /* new symbol tables are created with the current one as the     */
    /* parent.  Symbols can then be searched for in a hierarchical   */
    /* fashion.                                                      */
    /*---------------------------------------------------------------*/
    static private HglSymbolTable _globalSymbolTable = null;
    static private HglSymbolTable _childSymbolTable = null;
    static private HglSymbolTable _functionSymbolTable = null;

    // Keep track of which tree walker pass we're on.
    static private int _passNum = 0;
    static private boolean DEBUG=true;
}

/*-------------------------------------------------------------------*/
/* Simple declaration.  Note that because ANTLR will match something */
/* on a best-effort basis, those DECLs which have three children,    */
/* like arrays and functions, will still match this rule.            */
/*-------------------------------------------------------------------*/
declar throws HGLExceptionLine
{
    String name, argName;
    HglVariable funDecl, hvar;
    int j, numArgs;

    // Set current symbol table
    HglSymbolTable currTbl;
    if (_childSymbolTable != null)
      currTbl = _childSymbolTable;
    else
      currTbl = _globalSymbolTable;
}
    : #(DECL type:. i:ID (v:.)?)
        {
            /*-----------------------------------------------*/
            /* The 'v:." part will match anything.  We already   */
            /* know we have a valid type because the parser has  */
            /* completed successfully.                           */
            /* v is the optional third argument, and can be one    */
            /* of:                                        */
            /* 1) Array ("[x]")                              */
            /* 2) Function Declaration (prototype) (arglist;)   */
            /* 3) Function Definition  (arglist; cmpnd_stmt)     */
            /*-----------------------------------------------*/
            name = i.getText();

            if (v != null) {
                switch(v.getType()) {
                case FUNDECL:
                    // Only pick up functions on first pass
                    if (_passNum > 1) break;

                    // Make sure we're in global scope
                    if (_childSymbolTable != null) {
                        throw new HGLExceptionLine(i,
                            "Cannot use function declaration in this scope.");
                    } // if (_childSymbolTable != null)

                    HGLCommon.logger.finest("FUNDECL");
                    // Upon seeing decl list, create new symbol table to
                    // store in HglVariable::_value.
                    if (_functionSymbolTable.containsKey(name))
                    {
                        throw new HGLExceptionLine(i,
                            "Function " + name + " already defined");
                    }
                    funDecl = new HglVariable(type.getType()); // Return type

                    // Save PARM_LIST_TYPE tree, if any
                    if (v.getNumberOfChildren()>0) {
                        funDecl.setValue(v.getFirstChild());
                    } // if (v.getNumberOfChildren())
                    else {
                        funDecl.setValue(null);
                    } // else

                    _functionSymbolTable.put(name, funDecl);
                    break;
                case FUNDEF:
                {
                    // Only pick up functions on first pass
                    if (_passNum > 1) break;

                    // Make sure we're in global scope
                    if (_childSymbolTable != null) {
                        throw new HGLError(v,
                                "cannot place function declaration in this scope.",
                                HGLCommon.currentFileName);
                    } // if (_childSymbolTable != null)

                    HGLCommon.logger.finest("FUNDEF");

                    /*-----------------------------------------------------------
                     * Extract the agruments and the definition AST from the node.
                     */
                    AST args, definitionAST;
                    args = v.getFirstChild();
                    if (args.getType() == PARM_LIST_TYPE)
                    {
                        definitionAST = args.getNextSibling();
                    }
                    else
                    {
                        definitionAST = args;
                        args = null;
                    }

                    /*-----------------------------------------------------------------
                     * Check and see if this definition matches a pre-existing
                     * prototype.
                     */
                    funDecl = (HglVariable)_functionSymbolTable.get(name);

                    /*-----------------------------------------------------------------
                     * Name exists symbol table from
                     * declaration..check for redifinition
                     */
                    if (funDecl != null) {
                        // Verify this isn't a redefinition
                        AST declArgs;
                        HGLCommon.logger.finer("Found declaration "+i.getLine());
                        if (funDecl.isDefSet()) {
                            throw new HGLError(v,
                                "function " + name + " already defined",
                                HGLCommon.currentFileName);
                        }

                        // Make sure the parm list matches prototype
                        declArgs = (AST)funDecl.getValue();
                        if (((args != null) && (declArgs == null)) ||
                            ((args != null) && (declArgs != null) &&
                             ((args.getNumberOfChildren() !=
declArgs.getNumberOfChildren()) ||
                              (args.equalsTree(declArgs) == false))) ||
                            ((args == null) && (declArgs != null)))
                        {
                            throw new HGLExceptionLine(i,
                                name + " signature does not match declaration");
                        }
                    } // if (funDecl != null)
                    else {
                        HGLCommon.logger.finer("Did not find declaration "+i.getLine());

                        /*-----------------------------------------------------------------
-
                         * No existing prototype..create new entry in
_functionSymbolTable
                         */
                        funDecl = new HglVariable(type.getType()); // Return type
                        funDecl.setValue(args);
                        _functionSymbolTable.put(name, funDecl);
                    } // else

                    /*-----------------------------------------------------------------
                     * Save the implementation AST.
                     */
                    funDecl.setDefAST(definitionAST);

                    break;
                }
                case ARRAY_DECL:
                    if (_passNum == 1) break;
                    HGLCommon.logger.finer("ARRAY_DECL for " + name + "[" +
v.getFirstChild().getText() + "]");
                    // Array

                    if (currTbl.containsKey(name))
                    {
```

```
                    throw new HGLError(i, name + " already defined",
                        HGLCommon.currentFileName);
                }

                Object aSzExpr = expression(v.getFirstChild().getFirstChild());

                if ( !(aSzExpr instanceof Integer) ) {
                    throw new HGLError(i, "Array size must be integer expression",
                        HGLCommon.currentFileName);
                }

                int arraySize = ((Integer)aSzExpr).intValue();
                hvar = new HglVariable(type.getType(), arraySize);

                Object[] oArray = new Object[arraySize];
                hvar.setValue(oArray);

                try {
                    // Check if HTML variable and initialize it
                    if ((type.getType() > HTML_START) &&
                        (type.getType() < HTML_END))
                    {
                        Class c =  Class.forName(type.getText());
                        for (int x = 0; x < arraySize; x++)
                        {
                            oArray[x] = c.newInstance();
                        }
                    }
                } // try
                catch (Exception e) {
                    throw new HGLError(i, "Could not initialize HTML object " + name
+
                        ": " + e.getMessage(), HGLCommon.currentFileName);
                } // catch (Exception e)

                currTbl.put(name, hvar);

                break;
            } // switch(v.getType())
        } // if (v != null)
        else {
            if (_passNum != 1) {
                // Regular variable
                if (currTbl.containsKey(name))
                {
                    throw new HGLError(i, name + " already defined",
                        HGLCommon.currentFileName);
                }

                hvar = new HglVariable(type.getType());

                try {
                    // Check if HTML variable and initialize it
                    if ((type.getType() > HTML_START) &&
                        (type.getType() < HTML_END))
                    {
                        Class c =  Class.forName(type.getText());
                        hvar.setValue(c.newInstance());
                    }
                } // try
                catch (Exception e) {
                    throw new HGLError(i, "Could not initialize HTML object " + name
+
                        ": " + e.getMessage(), HGLCommon.currentFileName);
                } // catch (Exception e)

                currTbl.put(name, hvar);

            } // if (_passNum != 1)
        } // else
    }
    ;

/*-----------------------------------------------------------------*/
/* Statements.                                                     */
/*-----------------------------------------------------------------*/

compoundStatement throws HGLExceptionLine
{
    /*-------------------------------------------------------------
     * "push" new scope onto "stack"
     */
    HGLCommon.logger.finer("Pushing table for CSTMT");
    _childSymbolTable =
    new HglSymbolTable(
        ( _childSymbolTable!=null)
        ? _childSymbolTable
        : _globalSymbolTable,
        false);
}
    : #(CSTMT (declar | statement)*)
        {
            HGLCommon.logger.finer("Popping table for CSTMT");
            /*-----------------------------------------------
             * "pop" this scope off "stack"
             */
            if (_childSymbolTable.getParent() == _globalSymbolTable)
                _childSymbolTable = null;
            else
                _childSymbolTable = _childSymbolTable.getParent();
        }
    ;

statement throws HGLExceptionLine
{
    Object r = null;
}
    : #(STMT var:.)
        {
            if (_passNum != 1)
            {
                switch (var.getType()) {
                case EXPR:
                {
                    HGLCommon.logger.finer("Statement : expression");
                    try
                    {
                        r = expression(var.getFirstChild());
                    }
                    catch (Throwable t)
                    {
                        if (t instanceof HGLExceptionLine)
                            throw (HGLExceptionLine)t;
                        else
                            throw new HGLExceptionLine(var.getFirstChild(),
t.getMessage());
                    }
                    break;
                }
                case CSTMT:
                {
                    compoundStatement(var);
                    break;
                }
                case IF:
                {
                    HGLCommon.logger.finer("Statement : if");
                    AST conditional = var.getFirstChild();
                    HGLCommon.logger.finest("conditional type = " + conditional);
                    AST thenClause = conditional.getNextSibling();
                    HGLCommon.logger.finest("then type = " + thenClause);
                    AST elseClause = thenClause.getNextSibling();
                    HGLCommon.logger.finest("else type = " + elseClause);

                    Object condEval = expression(conditional.getFirstChild());
                    if ((condEval != null) && (condEval.equals(Boolean.TRUE)))
                    {
                        HGLCommon.logger.finest("true conditional");
                        statement(thenClause);
                    }
                    else
                    {
                        HGLCommon.logger.finest("false conditional");
                        if (elseClause != null)
                            statement(elseClause);
                    }
                    break;
                }
                case WHILE:
                {
                    HGLCommon.logger.finer("Statement : while");
                    AST conditional = var.getFirstChild();
                    AST loopPart = conditional.getNextSibling();
                    Object condEval = expression(conditional.getFirstChild());
                    while ((condEval != null) &&
                           (condEval.equals(Boolean.TRUE)))
                    {
                        boolean broken = false;

                        try
                        {
                            statement(loopPart);
                        }
                        catch (HGLInterruptFlowException ife)
                        {
                            /*-----------------------------------------------*/
                            /* A while loop can be broken by continue or     */
                            /* break.  If we encountered a return, bypass    */
                            /* this loop.                                    */
                            /*-----------------------------------------------*/
                            if (ife.isReturn()) throw ife;
                            else if (ife.isBreak())
                            {
                                /*-------------------------------------------*/
                                /* In the event of a break, we do not want  */
                                /* to evaluate the conditional again.       */
                                /*-------------------------------------------*/
                                broken = true;
                                condEval = Boolean.FALSE;
                            }
                        }
                        finally
                        {
                            if (!broken) condEval =
expression(conditional.getFirstChild());
                        }
                    }
                    break;
                }
                case RETURN:
                case BREAK:
                case CONTINUE:
                {
                    /*-------------------------------------------------------*/
                    /* These statement types result in flow of control       */
                    /* changing.                                             */
                    /*-------------------------------------------------------*/
                    HGLCommon.logger.finer("Statement : " +
                        HGLCommon.lexerTypeToString(var.getType()));
                    throw new HGLInterruptFlowException(var, this);
                }
                } // switch (var.getType())
            }
```

```
        }
    ;

/*------------------------------------------------------------------------------*/
/* Expressions.                                                                 */
/*------------------------------------------------------------------------------*/
expression returns [ Object r ] throws HGLExceptionLine
{
    Object o; r = null;
    Object a = null;
    Object b = null;
    TypeChecker checker = new TypeChecker();

    // Set current symbol table
    HglSymbolTable currTbl;
    if (_childSymbolTable != null)
      currTbl = _childSymbolTable;
    else
      currTbl = _globalSymbolTable;

    if (_passNum == 1) return null;

} : #(ASSIGN var:. o = expression) {
      String name = null;
      AST variable = var;
      int arrayIndex = 0;

      /*----------------------------------------------------------------------*/
      /* For HTML propertise, we throw it to FUNCTIONCALLSET rule.          *
      /*----------------------------------------------------------------------*/

      if (var.getType() == FUNCTIONCALLSET)
      {
         expression(var);
      } else {
      /*----------------------------------------------------------------------*/
      /* For arrays, the actual variable name is below the ARRAYDEREF item. */
      /*----------------------------------------------------------------------*/
      if (var.getType() == ARRAYDEREF)
      {
          variable = var.getFirstChild();

          //arrayIndex = Integer.parseInt(variable.getNextSibling().getText());
          Object arrayIndexObject = expression(variable.getNextSibling());
          if (arrayIndexObject instanceof Integer)
          {
              arrayIndex = ((Integer)arrayIndexObject).intValue();
          }
          else
          {
              HGLCommon.scold(new HGLExceptionLine(variable,
                              "Array index must be an integral type"));
          }
      }

      /*----------------------------------------------------------------------*/
      /* Strip the '$' off the variable name.                                 */
      /*----------------------------------------------------------------------*/
      name = variable.getText().substring(1);

      HglVariable hvar = (HglVariable)currTbl.get(name);

      if (hvar != null)
      {
          /*------------------------------------------------------------------*/
          /* Handle array initialization                                      */
          /*------------------------------------------------------------------*/
          if (o instanceof Object[])
          {
              Object[] arrayValues = (Object[])o;
              if ((hvar.isArray() == false) || (var.getType() == ARRAYDEREF))
              {
                  HGLCommon.scold(new HGLExceptionLine(variable,
                                  "Cannot initialize non-array variable " +
                                  "with the array initializer operator"));
              }
              else
              {
                  Object[] arrayStore = (Object[])hvar.getValue();
                  if ((arrayStore.length) != (arrayValues.length))
                  {
                      HGLCommon.scold(new HGLExceptionLine(variable,
                                      "Array initializer is not the same " +
                                      "length as the declared array"));
                  }
                  else
                  {
                      for (int x = 0; x < arrayStore.length; x++)
                      {
                          if (checker.validateType(hvar.getType(),
                                                   arrayValues[x]))
                          {
                              Object varToSet = arrayValues[x];

                              /*------------------------------------------*/
                              /* Assigning an Int to a Float is special.   */
                              /*------------------------------------------*/
                              if ((hvar.getType() == FLOAT) &&
                                  (varToSet instanceof Integer))
                              {
                                  varToSet = new Double(((Integer)varToSet).
                                                        doubleValue());
                              }

                              arrayStore[x] = varToSet;
                          }
                          else
                          {
                              String varType = HGLCommon.lexerTypeToString(
                                               hvar.getType());
                              HGLCommon.scold(new HGLExceptionLine(variable,
                                  "Cannot assign " + arrayValues[x] + " to " +
                                   varType + " " + name));
                          }
                      }
                  }
              }
          }
          /*-------------------------------------------------------------------*/
          /* Make sure that, if the left side is an array dereference, the  */
          /* variable is an array type, and vice versa.                      */
          /*-------------------------------------------------------------------*/
          else if (((hvar.isArray()) && (var.getType() != ARRAYDEREF)) ||
              ((hvar.isArray() == false) && (var.getType() == ARRAYDEREF)))
          {
              HGLCommon.scold(new HGLExceptionLine(
                variable, "Cannot assign between an array and a regular variable"));
          }
          else
          {
              Object[] arrayObject = null;

              if (hvar.isArray()) arrayObject = (Object[])hvar.getValue();

              if (checker.validateType(hvar.getType(), o))
              {
                  try
                  {
                      Object varToSet = o;

                      /*----------------------------------------------------*/
```

```
                        /* Assigning an Int to a Float is a special case.     */
                        /*----------------------------------------------------*/
                        if ((hvar.getType() == FLOAT) && (o instanceof Integer))
                        {
                            varToSet = new Double(((Integer)o).doubleValue());
                        }

                        /*----------------------------------------------------*/
                        /* The variable already references the array, so just */
                        /* set the array element.  For non-arrays, set the    */
                        /* value into the variable directly.                  */
                        /*----------------------------------------------------*/
                        if (arrayObject != null)
                        {
                            arrayObject[arrayIndex] = varToSet;
                            r = varToSet;
                        }
                        else
                        {
                            r = hvar.setValue(varToSet);
                        }
                    }
                    catch (ArrayIndexOutOfBoundsException aioobe)
                    {
                        HGLCommon.scold(new HGLExceptionLine(variable,
                            "Array index " + arrayIndex + " out of bounds"));
                    }
                }
                else
                {
                    String varType = HGLCommon.lexerTypeToString(hvar.getType());
                    HGLCommon.scold(new HGLExceptionLine(variable,
                                    "Cannot assign " + o + " to " +
                                     varType + " " + name));
                }
            }
        }
        else
        {
            HGLCommon.scold(new HGLExceptionLine(var, "Undeclared variable: " +
                                                      name));
        }
      }
    }
  | #(EXPR a=expression)
    {
          r = a;
          HGLCommon.logger.finest("Nested expression = " + r);
    }

  | #(STR_CAT a=expression b=expression)
    {
          r = checker.validateData(a, b, STR_CAT);
          HGLCommon.logger.finest("StrCat result = " + r);
    }
  | #(COND_OR a=expression b=expression)
    {
          r = checker.validateData(a, b, COND_OR);

          HGLCommon.logger.finest("Cond OR result = " + r);
    }

  | #(COND_AND a=expression b=expression)
    {
          r = checker.validateData(a, b, COND_AND);
          HGLCommon.logger.finest("Cond AND result = " + r);
    }

  | #(REL_EQ a=expression b=expression)
    {
          r = checker.validateData(a, b, REL_EQ);
```

```
            HGLCommon.logger.finest("Equals result = " + r);
    }


  | #(REL_NEQ a=expression b=expression)
    {
            r = checker.validateData(a, b, REL_NEQ);
            HGLCommon.logger.finest("NotEquals result = " + r);
    }

  | #(REL_LT {HGLCommon.logger.finest("Found a LT tree...");} a=expression b=expression)
    {
            r = checker.validateData(a, b, REL_LT);
            HGLCommon.logger.finest("LessThan result = " + r);
    }

  | #(REL_LTE a=expression b=expression)
    {
            r = checker.validateData(a, b, REL_LTE);
            HGLCommon.logger.finest("LessThanOrEquals result = " + r);
    }

  | #(REL_GT { HGLCommon.logger.finest("Found a GT tree..."); } a=expression b=expression)
    {
            r = checker.validateData(a, b, REL_GT);
            HGLCommon.logger.finest("GreaterThan result = " + r);
    }

  | #(REL_GTE a=expression b=expression)
    {
            r = checker.validateData(a, b, REL_GTE);
            HGLCommon.logger.finest("GreaterThanOrEqual result = " + r);
    }

  | ( #(PLUS a=expression b=expression) ) => #(PLUS a=expression b=expression)
    {
        HGLCommon.logger.finest("Trying addition rule... " + a + " plus " + b);
        r = checker.validateData(a, b, PLUS);
        HGLCommon.logger.finer("Addition result = " + r);
    }

  | #(PLUS a=expression )
    {
          HGLCommon.logger.info("Trying unary positive... ");
          r = checker.validateData(a, null, UPLUS);
          HGLCommon.logger.finest("Unary positive = " + r);
     }

  | ( #(MINUS a=expression b=expression) ) => #(MINUS a=expression b=expression)
    {
            HGLCommon.logger.info("Trying subtraction... ");
            r = checker.validateData(a, b, MINUS);
            HGLCommon.logger.finest("Subtraction result = " + r);
    }

  | #(MINUS a=expression )
    {
          HGLCommon.logger.info("Trying unary negation... ");
          r = checker.validateData(a, null, UMINUS);
          HGLCommon.logger.finest("Unary negation = " + r);
     }

  | #(TIMES a=expression b=expression)
    {
            r = checker.validateData(a, b, TIMES);
            HGLCommon.logger.finest("Multiplication result = " + r);
    }

  | #(DIV a=expression b=expression)
    {
            r = checker.validateData(a, b, DIV);
    }
      | #(FUNCTIONCALL fn:ID (p:PARM_LIST)?)
          {
              String fun_name = fn.getText();
              HGLCommon.logger.finer(" Function " + fun_name +
                  " called on line " + fn.getLine());

              if (fun_name.equals("arraySize")) {

                  AST arrayVarNode = p.getFirstChild();
                  arrayVarNode = arrayVarNode.getFirstChild();

                  if (arrayVarNode == null) {
                      HGLCommon.scold(new HGLExceptionLine(fn,
                              "Must supply array argument."));
                  } // if (arrayVarNode == null)

                  if ( arrayVarNode.getType() != VARIABLE ) {
                      HGLCommon.scold(new HGLExceptionLine(fn,
                              "Argument has incorrect type."));
                  } // if ( arrayVarNode.getType() != VARIABLE )

                  HglVariable arrayVar =
(HglVariable)currTbl.get(arrayVarNode.getText().substring(1));

                  if ( arrayVar == null ) {
                      HGLCommon.scold(new HGLExceptionLine(fn,
                              "Undefined variable."));
                  } // if ( arrayVar == null )

                  Object curActualParmValue = null;
                  if ( !arrayVar.isArray() ) {
                      HGLCommon.scold(new HGLExceptionLine(fn,
                              "Argument is not an array."));
                  } // if ( !arrayVar.isArray() )

                  curActualParmValue = arrayVar.getValue();
                  r = new Integer(((Object[])curActualParmValue).length);
              } else {

                  HglVariable funDef =
                  (HglVariable)_functionSymbolTable.get(fun_name);

                  if (funDef == null)
                  throw new RuntimeException("Undeclared function " + fun_name);
                  else
                  {
                      /*--------------------------------------------------------
                   * Has this function been defined?
                   */
                      if (!funDef.isDefSet())
                      throw new RuntimeException("Function " + fun_name +
                          " not defined.");

                      HGLCommon.logger.finer("Function " + fun_name + "found");

                      /*--------------------------------------------------------
                   * Validate argument number/type.
                   */
                      HglSymbolTable fSymTable = HGLCommon.parseFunctionParms(
                          (AST)funDef.getValue(), p, _globalSymbolTable, this);
                      HglSymbolTable oldTable = _childSymbolTable;
                      _childSymbolTable = fSymTable;

                      /*--------------------------------------------------------
                   * Call compoundStatement on function tree.
                   */
                      try
                      {
                          compoundStatement((AST)funDef.getDefAST());
                      }
```

```
                    catch (HGLInterruptFlowException ife)
                    {
                        /*-----------------------------------------------------*/
                        /* A return statement was encountered.  Lets see if it  */
                        /* is valid.                                            */
                        /*-----------------------------------------------------*/
                        if (ife.isReturn())
                        {
                            Object retObj = ife.getReturnValue();
                            int fcnReturnType = funDef.getType();

                            if (checker.validateType(fcnReturnType, retObj))
                            {
                                r = retObj;
                            }
                            else
                            {
                                HGLCommon.scold(new HGLExceptionLine(ife.getAST(),
                                        "Cannot return " + retObj + " to function of type
" +
                                        HGLCommon.lexerTypeToString(fcnReturnType)));
                            }
                        }
                        else
                        {
                            /*-------------------------------------------------*/
                            /* We got here because of a break or a continue.    */
                            /* This is invalid, we expected a return statement. */
                            /*-------------------------------------------------*/
                            AST bStatement = ife.getAST();
                            String sType = HGLCommon.lexerTypeToString(
                                bStatement.getType());
                            HGLCommon.scold(new HGLExceptionLine(bStatement,
                                    "Cannot end function with " + sType + " statement"));
                        }
                    }
                    finally
                    {
                        _childSymbolTable = oldTable;

                        /*-----------------------------------------------------*/
                        /* Make sure that a return statement was encountered    */
                        /* for this invocation of the function.  Since the file */
                        /* is interpreted, we only detect returns when the      */
                        /* function is actually invoked.  One invocation may be */
                        /* able to return successfully while others may not.    */
                        /*-----------------------------------------------------*/
                        if (r == null) /* TODO: && (funDef.getType() != VOID) */
                        {
                            HGLCommon.scold(new HGLExceptionLine(fn,
/*(AST)funDef.getDefAST(),*/
                                    "Function invocation had no return value"));
                        }
                    }
                } // else
            } // } else
        }
    | #(FUNCTIONCALLOBJ a=expression method:PROPID (pl:PARM_LIST)?)
        {

            String fun_name = method.getText().substring(1);

            if (a == null)
               throw new HGLExceptionLine(method, "Undeclared HTML object");

            if (!(a instanceof HglHTMLObject))
              throw new HGLExceptionLine(method, "Variable is not an HTML object.");

            HGLCommon.logger.finer("Method " + fun_name
                + " at line " + method.getLine());
            try {
                HGLCommon.runWebFun(this, (HglHTMLObject)a, fun_name, pl);
            }
            catch (HGLPropertyException e) {
                HGLCommon.scold(new HGLWarning(method,
                        e.getPropName() + " is an invalid property for this type.",
                        HGLCommon.currentFileName));
            }
            catch (HGLException e) {
                HGLCommon.scold(new HGLError(method,
                        e.getMessage(), HGLCommon.currentFileName));
            }

            r = a;
        }

    | #(fs:FUNCTIONCALLSET a=expression property:PROPID)

        {
            String prop_name = property.getText().substring(1);

            HGLCommon.logger.info(prop_name + ": assignment syntax for set_html called on
" +

                prop_name + " at line: " + property.getLine());

            if (a == null)
               throw new HGLExceptionLine(property, "Undeclared HTML object");

            if (!(a instanceof HglHTMLObject))
              throw new HGLExceptionLine(property, "Variable is not an HTML object.");

            try {
                ((HglHTMLObject)a).html_set(prop_name, expression(fs.getNextSibling()));
            }
            catch (HGLPropertyException pe) {
                HGLCommon.scold(new HGLWarning(property,
                        pe.getPropName() + " is an invalid property for this type.",
                        HGLCommon.currentFileName));
            }
        }
    | aInit:ARRAYINIT
  {
      int numElems = aInit.getNumberOfChildren();

      Object[] objects = new Object[numElems];
      AST curObject = aInit.getFirstChild();

      for (int x = 0; x < numElems; x++)
      {
          objects[x] = expression(curObject);
          curObject = curObject.getNextSibling();
      }

      r = objects;
  }
| aDeref:ARRAYDEREF
  {
      AST var1 = aDeref.getFirstChild();

      //int arrayIndex = Integer.parseInt(var1.getNextSibling().getText());
      int arrayIndex = 0;
      Object arrayIndexObject = expression(var1.getNextSibling());
      if (arrayIndexObject instanceof Integer)
      {
          arrayIndex = ((Integer)arrayIndexObject).intValue();
      }
      else
      {
          HGLCommon.scold(new HGLExceptionLine(var1,
                          "Array index must be an integral type"));
      }
```

```
        /*-----------------------------------------------------*/
        /* Strip the '$' off the variable name.                */
        /*-----------------------------------------------------*/
        String name = var1.getText().substring(1);
        HGLCommon.logger.finer("expression::ARRAYDEREF=" + name + arrayIndex);
        HglVariable hvar = (HglVariable)currTbl.get(name);

        if (hvar == null)
        {
            HGLCommon.scold(new HGLExceptionLine(
                            var1, "Undefined variable " + name));
        }
        else if (hvar.isArray() == false)
        {
            HGLCommon.scold(new HGLExceptionLine(
                            var1, name + " is not an array"));
        }
        else
        {
            Object[] arrayValues = (Object[])hvar.getValue();
            try
            {
                r = arrayValues[arrayIndex];

                if (r == null)
                {
                    HGLCommon.scold(new HGLExceptionLine(var1,
                                    "Uninitialized array index " + arrayIndex));
                }
            }
            catch (ArrayIndexOutOfBoundsException aioobe)
            {
                HGLCommon.scold(new HGLExceptionLine(
                                var1, "Invalid array index " + arrayIndex));
            }
        }
    }
  | v:VARIABLE
    {
        /*-----------------------------------------------------*/
        /* Strip the '$' off the variable name.                */
        /*-----------------------------------------------------*/
        String name = v.getText().substring(1);
        HGLCommon.logger.finer("expression::VARIABLE=" + name);
        HglVariable hvar = (HglVariable)currTbl.get(name);

        if (hvar != null)
        {
            if (hvar.isArray())
            {
                HGLCommon.scold(new HGLExceptionLine(v,
                                "Cannot return array as a primitive type"));
            }
            else
            {
                r = hvar.getValue();

                if (r == null)
                {
                    HGLCommon.scold(new HGLExceptionLine(v,
                                    "Uninitialized variable " + name));
                }
            }
        }
        else
        {
            HGLCommon.scold(new HGLExceptionLine(v,
                            "Undeclared variable: " + name));
        }
    }
```

```
| i:INT_LIT { r = new Integer(i.getText()); }
| f:FLOAT_LIT {r = new Double(f.getText());}
| c:CHAR_LIT { r = new Character(c.getText().charAt(0)); }
| str:STRING { r = str.getText();}
| tLit:TRUE { r = Boolean.TRUE; }
| fLit:FALSE { r = Boolean.FALSE; }
| n:NULL{r = null;}
;
/*-------------------------------------------------------------------*/
/* Start rule is the root of the tree.                               */
/*-------------------------------------------------------------------*/
start throws HGLExceptionLine
{
    Object o;
}
    : #(START
            {
                // value after increment: 1=first pass 2=second pass
                _passNum++;

                /*-----------------------------------------------*/
                /* Initialization performed once before any other    */
                /* rules are processed.                              */
                /*-----------------------------------------------*/
                if (_passNum == 1) {
                    _globalSymbolTable  = new HglSymbolTable(null, false);
                    _functionSymbolTable = new HglSymbolTable(null, false);
                }
                HGLCommon.logger.finer("On pass #" + _passNum);
            }
            (declar | statement)*
        )
    ;
```

10.2.2 HGL Exceptions

```
--------------------
---- ./HGLError.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLError.java
// Purpose: Fatal or near-fatal run-time exception with line-number
//          information in the AST node.
//////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLError extends HGLExceptionLine
{
   public HGLError(AST t, String message, String fileName) {
    super(t, message, fileName);
   }
}
--------------------
---- ./HGLException.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLException.java
// Purpose: Exception thrown only by HGL.
//////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLException extends Exception
 {
   public HGLException(String message) { super(message); }
 }
--------------------
```

```
---- ./HGLExceptionLine.java
Author: Yan Koyfman
--------------------
////////////////////////////////////////////////////////////////////
// Module: HGLExceptionLine.java
// Purpose: Language exception with line-number and file info.
////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLExceptionLine extends HGLException
{

   AST _t;
   String _fileName;

   /**
   * Constructor.
   */
   public HGLExceptionLine(AST t, String message) {
    super(message);
    _t = t;
    _fileName = HGLCommon.currentFileName;
   }

   /**
   * Constructor.
   */
   public HGLExceptionLine(AST t, String message, String fileName) {
    super(message);
    _t = t;
    _fileName = fileName;
   }

   int getLine() {
    return _t.getLine();
   } // int getLine()


   String getFileName() {
    return _fileName;
   } // int getLine()
 }

--------------------
---- ./HGLInterruptFlowException.java
Author: Tim Kaczynski
--------------------
////////////////////////////////////////////////////////////////////
// Module: HGLInterruptFlowException.java
// Purpose: Language exception to break/continue/return in a
//          compound statement
////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLInterruptFlowException extends HGLExceptionLine
    implements HGLLexerTokenTypes
{
    /**
     * The AST for the breaking statement
     */
    private AST _breakingStatement;


    /**
     * The tree walker for the AST
     */
    private HGLTreeWalker _walker;


    /**
     * Constructor
     */
    public HGLInterruptFlowException(AST t, HGLTreeWalker walker)
    {
        super(t, "Unexpected statement of type " +
                 HGLCommon.lexerTypeToString(t.getType()));

        _breakingStatement = t;
        _walker = walker;
    }


    /**
     * Gets the return value for return statements
     */
    public Object getReturnValue() throws HGLExceptionLine
    {
        Object returnObj = null;

        if (isReturn())
        {
            try
            {
                returnObj =
                    _walker.expression(_breakingStatement.getFirstChild());
            }
            catch (Throwable t)
            {
                throw new HGLExceptionLine(_breakingStatement.getFirstChild(),
                                           "Could not parse return value");
            }
        }

        return returnObj;
    }


    /**
     * Tells whether this is a break from a return statement
     */
    public boolean isReturn()
    {
        return (_breakingStatement.getType() == RETURN);
    }


    /**
     * Tells whether this is a break from a break statement
     */
    public boolean isBreak()
    {
        return (_breakingStatement.getType() == BREAK);
    }


    /**
     * Tells whether this is a break from a continue statement
     */
    public boolean isContinue()
    {
        return (_breakingStatement.getType() == CONTINUE);
    }


    /**
     * Gets the AST that interrupted the flow
     */
    public AST getAST()
    {
        return _breakingStatement;
    }
}
```

```
--------------------
---- ./HGLPropertyException.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLPropertyException.java
// Purpose: Invalid property run-time exception.
//////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLPropertyException extends HGLException
 {

   String _name;

   /**
   * Constructor.
   */
   public HGLPropertyException(String name) {
    super(name);
    _name = name;
   }

   public String getPropName() {
    return _name ;
   } // int getPropName()

 }
--------------------
---- ./HGLWarning.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLWarning.java
// Purpose: Non-fatal run-time exception with line-number information
//          in the AST node.
//////////////////////////////////////////////////////////////////////
import antlr.collections.*;

public class HGLWarning extends HGLExceptionLine
{
   public HGLWarning(AST t, String message, String fileName) {
    super(t, message, fileName);
   }
}

--------------------
---- ./InvalidHGLTypeException.java
Author: Tim Kaczynski
--------------------
/**
 * Invalid type exception for HGL.
 */
 public class InvalidHGLTypeException extends Exception
 {
     /**
      * Constructor.
      */
     public InvalidHGLTypeException()
     {
        super();
     }

     /**
      * Constructor.
      */
     public InvalidHGLTypeException(String message)
     {
        super(message);
     }
}
```

10.2.3 HGL Logging and Related Classes

```
--------------------
---- ./HGLLogFormatter.java
Author: Yan Koyfman
--------------------
import java.util.logging.*;
import java.util.regex.*;

class HGLLogFormatter extends Formatter {

  private boolean enabled = true;
  private boolean cls = false;
  private boolean fun = false;

  public HGLLogFormatter(Object options) {
   if (options instanceof Boolean) {
     // Turn off all tracing
     if (options == Boolean.FALSE) {
      enabled = false;
      HGLCommon.logger.setLevel(Level.OFF);
      return;
     } // if (options == Boolean.FALSE)

     // Default "on" behavior
     if (options == Boolean.TRUE)
      return;
   } // if (options instanceof Boolean)
   else {
     if (((String)options).matches(".*c.*"))
      cls = true;
     if (((String)options).matches(".*f.*"))
      fun = true;
   } // else
  }

  public String format(LogRecord r)
  {

   if ( !enabled ) return null;

   String trace = "";

   if (cls) trace += r.getSourceClassName() + "::";
   if (fun) trace += r.getSourceMethodName() + "()";

   trace += " " + r.getMessage() + "\n";

   return trace;
  }
}

--------------------
---- ./ASTLines.java
Author: Shruti Gandhi
--------------------
import antlr.CommonAST;
import antlr.Token;
public class ASTLines extends CommonAST {
  private int line = 0;

  java.util.logging.Logger logger = java.util.logging.Logger.getLogger("hglLog");
  public void initialize(Token tok) {
   super.initialize(tok);
   line=tok.getLine();
  }
  public int getLine(){
   return line;
  }
}
```

10.2.4 HGL HTML Objects, CSS Generation and Related Code

```
--------------------
---- ./HGLCommon.java
Author: Yan Koyfman & Tim Kaczynski
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLCommon.java
// Purpose: Shared code.
//////////////////////////////////////////////////////////////////////
import java.lang.reflect.*;
import antlr.collections.*;

class HGLCommon implements HGLLexerTokenTypes{
  public static final String funPrefix = "html_";
  public static boolean errorCondition = false;
  public static final java.util.logging.Logger logger
   = java.util.logging.Logger.getLogger("hglLog");

  public static String currentFileName;

  // Fill tbl with public HTML methods of class c
  public static void buildMethodTable(HglSymbolTable tbl, Class c) {
   int i;

   Method m[] = c.getMethods();

   for(i=0; i<m.length; i++) {
     if (m[i].getName().startsWith(funPrefix)) {
      logger.finer("Adding " + m[i].getName() + " to symtbl");
      tbl.put(m[i].getName(), m[i]);
     } // if (m[i].getName().startsWith(HGLCommon.funPrefix));
   }
  } // public static buildMethodTable(HglSymbolTable tbl, Class c)

  // Run function 'fun' on type 'c' for object in 'v'
  public static void runWebFun(HGLTreeWalker T,
                       HglHTMLObject v,
                       String fun,
                       AST pl) throws HGLPropertyException, HGLException {

   Object[] args;                      // Arguments to function
   Method m;                     // Method to call
   Object o;
   String methods = "methods";

   try {
     Class c = v.getClass();

     // Verify that this valid function has been invoked for this type.
     if ( (m = (Method) v.getWebMethods().get(funPrefix+fun)) == null ) {
      throw new HGLException("Method " + fun +
                  " not defined for" + c.getName() +
                  " objects.");
     } else {
      // Verify number and type of arguments.
      Class parms[] = m.getParameterTypes();

      // Permit two cases:
      // - no args are given and none expected
      // - number of given args is correct

      if ((pl == null) && (parms.length == 0))
```

```
        args = null;
      else if ((pl != null) && (parms.length == pl.getNumberOfChildren())) {
        args = new Object[parms.length];

        // First arg
        AST arg = pl.getFirstChild();
        o = T.expression(arg.getFirstChild());

        if (parms[0].isAssignableFrom(o.getClass())) {
         logger.finest("Assigning a " + o.getClass().getName() + " for a " +
parms[0].getName());
         args[0] = o;
         logger.finest("The argument is now a " + args[0].getClass().getName());

         // Subsequent args
         for (int j=1; j<parms.length; j++) {
           arg = arg.getNextSibling();
           o = T.expression(arg.getFirstChild());
           if (parms[j].isAssignableFrom(o.getClass())) {
            logger.finest("Assigning a " + o.getClass().getName() + " for a " +
parms[j].getName());
            args[j] = o;
            logger.finest("The argument is now a " + args[j].getClass().getName());
           } // if (parms[j].isAssignableFrom(o.getClass()))
           else
            throw new RuntimeException("Argument types do not match.");
         } // for (int j=1; j<parms.length; j++)
        } // if (parms[0].getName() == o.getName())
        else
         throw new RuntimeException("Argument types do not match.");
      } // else if ((pl != null) && (parms.length == pl.getNumberOfChildren()))
      else
        throw new RuntimeException("Wrong number of arguments.");

      // Invoke requested method on this object
      logger.finest("About to invoke method on.. " + v.getClass().getName());
      m.invoke(v, args);
     } // } else
    } // try
    catch (HGLException he) {
      throw he;
    }
    catch (Exception e) {
      if (e instanceof InvocationTargetException) {
       Throwable t = e.getCause();
       if (t instanceof HGLPropertyException)
         throw (HGLPropertyException)t;
       if (t instanceof HGLException)
         throw (HGLException)t;
       else {
         // Other kind of exception
         System.err.println(e.getCause().getMessage());
         e.getCause().printStackTrace();
       } // else
      } // if (e instanceof InvocationTargetException)
      else {
       throw new RuntimeException("Exception caught at invocation: "
                     + e.getMessage());
      } // else
    } // catch (Exception e)
   } // public static void runWebFun(HglVariable v, Class c, String fun)


     /**
      * Converts a lexer token type to a string.
      * @param type The lexer token type constant
      * @return The string representation of the lexer token type
      */
     public static String lexerTypeToString(int type)
     {
         String name = "Unknown";
         switch (type)
         {
             case (INTEGER):
                 name = "Integer";
                 break;
             case (FLOAT):
                 name = "Float";
                 break;
             case (CHAR):
                 name = "Char";
                 break;
             case (BOOLEAN):
                 name = "Boolean";
                 break;
             case (STR):
                 name = "String";
                 break;
             case (PARAGRAPH):
                 name = "Paragraph";
                 break;
             case (OLIST):
                 name = "Ordered List";
                 break;
             case (ULIST):
                 name = "Unordered List";
                 break;
             case (PAGE):
                 name = "Page";
                 break;
             case (BREAK):
                 name = "Break";
                 break;
             case (CONTINUE):
                 name = "Continue";
                 break;
             case (RETURN):
                 name = "Return";
                 break;
         }

         return name;
     }

     /**
      * Function converts the defined parameter list AST and real parameter
      * list for a function call into a symbol table.
      * @param definition The defined parameter list from the function
      *        definition or prototype
      * @param parameters The actual parameters supplied on the function
      *        call
      * @param parentTable The parent symbol table, which will be linked to
      *        the new symbol table.
      * @param walker The tree walker for this source file
      * @return An HglSymbolTable containing symbols for each parameter
      * @throws HGLExceptionLine thrown if an exception is encountered while
      *         walking a nested expression.
      */
     public static HglSymbolTable parseFunctionParms(
             AST definition, AST parameters, HglSymbolTable parentTable,
             HGLTreeWalker walker)
         throws HGLExceptionLine
     {
         logger.finer("parseFunctionParms entry");

         HglSymbolTable fSymTable = new HglSymbolTable(parentTable, false);

         /*------------------------------------------------------------------*/
         /* When both the caller and the definition have no parameters, we   */
         /* are done.                                                        */
         /*------------------------------------------------------------------*/
```

```
if ((definition == null) && (parameters == null))
{
    logger.finest("No parameters");
}
/*------------------------------------------------------------------*/
/* If one or the other is null, we have a problem.                 */
/*------------------------------------------------------------------*/
else if ((definition == null) || (parameters == null))
{
    throw new RuntimeException("Invalid (null) function parameters");
}
/*------------------------------------------------------------------*/
/* Otherwise, attempt to parse the arguments.                       */
/*------------------------------------------------------------------*/
else
{
    TypeChecker checker = new TypeChecker();

    AST curDefinedParm = definition.getFirstChild();
    int curDefinedParmType = curDefinedParm.getType();
    curDefinedParm = curDefinedParm.getNextSibling();
    String curDefinedParmName = curDefinedParm.getText();

    AST curActualParm = parameters.getFirstChild();
    Object curActualParmValue = null;
    try
    {
        curActualParmValue = walker.expression(curActualParm);
    }
    catch (antlr.RecognitionException e)
    {
        throw new RuntimeException("Error evaluating function " +
                                   "parameter " + curDefinedParmName);
    }

    while ((curDefinedParm != null) && (curActualParm != null))
    {
        /*----------------------------------------------------------*/
        /* Check to see if the parameter types match.               */
        /*----------------------------------------------------------*/
        if (checker.validateType(curDefinedParmType,
                                 curActualParmValue))
        {
            HglVariable var = new HglVariable(curDefinedParmType);

            /*------------------------------------------------------*/
            /* Assigning Int to Float is a special case.            */
            /*------------------------------------------------------*/
            if ((curDefinedParmType == FLOAT) &&
                (curActualParmValue instanceof Integer))
            {
                var.setValue(new Double(((Integer)curActualParmValue).
                                       doubleValue()));
            }
            else
            {
                var.setValue(curActualParmValue);
            }

            fSymTable.put(curDefinedParmName, var);
        }
        else
        {
            String targetTypeString =
                lexerTypeToString(curDefinedParmType);
            throw new RuntimeException(
                "Could not assign argument " + targetTypeString +
                " " + curDefinedParmName + " to " +
                curActualParmValue);
        }
```

```
                /*----------------------------------------------------------*/
                /* Advance to the next parameter (if one exists).           */
                /*----------------------------------------------------------*/
                curDefinedParm = curDefinedParm.getNextSibling();
                if (curDefinedParm != null)
                {
                    curDefinedParmType = curDefinedParm.getType();
                    curDefinedParm = curDefinedParm.getNextSibling();
                    curDefinedParmName = curDefinedParm.getText();
                }

                curActualParm = curActualParm.getNextSibling();
                if ((curActualParm != null) && (curDefinedParm != null))
                {
                    try
                    {
                        curActualParmValue = walker.expression(curActualParm);
                    }
                    catch (antlr.RecognitionException e)
                    {
                        throw new RuntimeException(
                            "Error evaluating function parameter " +
                            curDefinedParmName);
                    }

                }
            }

            /*--------------------------------------------------------------*/
            /* If one or the other (defined/actual) is null, but not both, */
            /* the parameter lists were not balanced.                       */
            /*--------------------------------------------------------------*/
            if (((curDefinedParm != null) && (curActualParm == null)) ||
                ((curDefinedParm == null) && (curActualParm != null)))
            {
                throw new RuntimeException("Function parameter lists are " +
                                           "incorrect length");
            }
        }

        logger.finer("parseFunctionParms exit");

        return fSymTable;
    }

  // Display an HGLWarning message
  public static void scold(HGLExceptionLine w) {
    String type = "Error: ";

    if (w instanceof HGLWarning)
      type = "Warning: ";
   else
     errorCondition = true;

    System.err.println(type + w.getFileName()+":"+w.getLine()+": "+w.getMessage());
  }

  // Display an HGLWarning message
  public static void scold(antlr.RecognitionException w) {
    String type = "Error: ";
    errorCondition = true;

    System.err.println(type + currentFileName+":"+w.getLine()+": "+w.getMessage());
  }

} // class HGLCommon


--------------------
---- ./HGLImage.java
Author: Yan Koyfman
```

```
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLImage.java
// Purpose: Implementation of the HTML image object.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;

class Image extends HglHTMLObject {

  private static String objType = "img";
  private static HglSymbolTable methods
   = new HglSymbolTable(null, false);
  private static int cssCnt = 0;

  /* Constructor */
  public Image() {
    super(methods);

    // Class specific properties
    nonCssProps.put("src", new HGLProperty("src", getClass().getName(), "src", true,
nonCssProperty));
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }
  public boolean hasCloseTag() { return false; }

  // Set URL for image
  public void html_setPath(String s) {
    try {
      html_set("src", s);
    }
    catch (HGLPropertyException he) {
      // Will not happen
    }
  }

  // Cannot add stuff
  public void html_add(Object o) throws HGLException {
    throw new HGLException("Cannot add objects to Images.");
  }

} // class Image


--------------------
---- ./HGLLink.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLLink.java
// Purpose: Implementation of the HTML link object.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;

class Link extends HglHTMLObject {

  private static String objType = "a";
  private static HglSymbolTable methods
   = new HglSymbolTable(null, false);
  private static int cssCnt = 0;

  /* Constructor */
  public Link() {
    super(methods);

    // Class specific properties
    nonCssProps.put("addr", new HGLProperty("addr", getClass().getName(), "href", true,
nonCssProperty));
```

```
   pseudoClasses.put("link", null);
   pseudoClasses.put("visited", null);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

} // class Link



--------------------
---- ./HGLListItem.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLListItem.java
// Purpose: A list element.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;

class ListItem extends HglHTMLObject {

  private static String objType = "li";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);

  private static int cssCnt = 0;

  /* Constructor */
  public ListItem(Object o) {
   super(methods);
   objs.add(o);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }
} // class ListItem



--------------------
---- ./HGLOList.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLOList.java
// Purpose: Ordered list HTML element.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;

class OList extends HglHTMLObject {

  private static String objType = "ol";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);

  private static int cssCnt = 0;

  /* Constructor */
  public OList() {
   super(methods);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

  public void html_addItem(Object o) throws HGLException {
   objs.add(new ListItem(o));
   HGLCommon.logger.finest("OList: html_addItem called");
  } // public void html_add(HglHTMLObject o)

  // Synonym for addItem
  public void html_add(Object o) throws HGLException {
   html_addItem(o);
   HGLCommon.logger.finest("OList: html_add called");
  } // public void html_add(HglHTMLObject o)

} // class OList


--------------------
---- ./HGLPage.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLPage.java
// Purpose: Implementation of the HTML page object.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;
import java.util.Iterator;

class Page extends HglHTMLObject {

  private static String objType = "body";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);
  private String _title = null;

  /* Constructor */
  public Page() {
   super(methods);
  }

  private void initProps() {
    // Empty for page
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }

  /*----------------------------------------------------------------
   * Display HTML features
   */
  public void html_print() {
   HGLCommon.logger.finest("html_print called");
   if (HGLCommon.errorCondition) return;

   // Remove mappings from CSS map
   cssClasses.clear();

   String css = "<html><head>\n";

   // Style sheet open tag
   css += "<style type=\"text/css\" title=\"hglCSS\" media=\"all\">\n";

   // Style for this page object and sub-objects
   css += style();

   // End tag for style
   css += "</style>\n";

   if (_title != null)
```

```
     css += "<title>"+_title+"</title>\n";

    // All the CSS code associated with this document
    hgl.pOut.println(css + "\n</head>\n");

    // The HTML of this document
    hgl.pOut.println(toString() + "\n</html>");

  } // public void html_print()

  public void html_setTitle(String t) {
   _title=t;
  } // public void html_setTitle(String t)

  // No CSS data for page..for now.
  protected int getCssCnt() { return 0; }
  protected void incCssCnt() {  }

} // class Page

--------------------
---- ./HGLParagraph.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLParagraph.java
// Purpose: Implementation of the HTML paragraph object.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;

class Paragraph extends HglHTMLObject {

  private static String objType = "p";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);
  private static int cssCnt = 0;

  /* Constructor */
  public Paragraph() {
    super(methods);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

} // class Paragraph

--------------------
---- ./HGLProperty.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLProperty.java
// Purpose: Property of an HTML object.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;

class HGLProperty {

  private Object value;
  private String name;
  private boolean quote;
  private boolean _nonCss = false;
  private static Vector cssProps = new Vector();
  private static Vector nonCssProps = new Vector();

  // Constructor for CSS properties
  public HGLProperty(String usr, String owner, String n, boolean q) {
    name = n;
    quote = q;
    cssProps.add(owner + ": " + usr + ": [CSS] " + n);
  } // public HGLProperty(String n, boolean q)

  // Constructor for non CSS properties
  public HGLProperty(String usr, String owner, String n, boolean q, boolean nonCss) {
    name = n;
    quote = q;
    _nonCss = nonCss;
    nonCssProps.add(owner + ": " + usr + ": [TAG] " + n);
  } // public HGLProperty(String n, boolean q)

  // Copy constructor that optionally clears value
  public HGLProperty(HGLProperty p, boolean clear) {
    name = p.name;
    quote = p.quote;
    _nonCss = p._nonCss;

   if (clear)
     value = null;
   else
     value = p.value;

  } // public HGLProperty(String n, boolean q)

  // A stringified value to print
  public String toString() {
    String eq;
    String ret = new String();

    if (_nonCss)
      eq = "=";
    else
      eq = ": ";

    if (quote)
      ret += name + eq + "\"" + value.toString() + "\"";
    else
      ret += name + eq + value.toString();

    if (!_nonCss)
      return ret + ";";
    else
      return ret;

  } // public getValue()

  public void setValue(Object v) {
    value = v;
  } // public setValue(Object v)

  public boolean isSet() { return value != null; }
  public static Vector getCssProps() { return cssProps; }
  public static Vector getNonCssProps() { return nonCssProps; }
  public Object getValue() { return value; }

} // class HGLProperty


--------------------
---- ./HGLSpan.java
Author: Yan Koyfman
--------------------
////////////////////////////////////////////////////////////////////
// Module: HGLSpan.java
// Purpose: Implementation of the HTML Span object.
////////////////////////////////////////////////////////////////////

class Span extends HglHTMLObject {

  private static String objType = "span";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);
  private static int cssCnt = 0;

  /* Constructor */
  public Span() {
    super(methods);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

} // class Span

--------------------
---- ./HGLTable.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLTable.java
// Purpose: Table HTML element.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;

class Table extends HglHTMLObject {
  private static String objType = "table";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);
  private static int cssCnt = 0;

  private HashMap tableRows =  new HashMap();

  // How big is our table?
  private int highRow = 0;
  private int highCol = 0;

  /* Constructor */
  public Table() {
    super(methods);
  }

  public void html_addElement(Object o, Integer row, Integer col) throws HGLException {

    HashMap r;
    TableCell cell;

    // Check if hash table for this row exists
    if ( (r = (HashMap)tableRows.get(row)) != null )
      if ( (cell = (TableCell)r.get(col)) != null )
   // Cell already exists in table; add new object to it
   cell.html_add(o);
      else {
   // Row exists, but no cell; add cell to row
   cell = new TableCell(o, row.intValue(), col.intValue());
   r.put(col, cell);
   // Stick ref objs vector for style printing
   objs.add(cell);
      }
    else {
      // No row or cell
      r = new HashMap();   // create row
      tableRows.put(row, r);
      cell = new TableCell(o, row.intValue(), col.intValue());
      r.put(col, cell);    // add cell to row
      // Stick ref objs vector for style printing
      objs.add(cell);
    }
    // Increase table size, if necassary.
    if (row.intValue() > highRow) highRow = row.intValue();
    if (col.intValue() > highCol) highCol = col.intValue();

    HGLCommon.logger.finest("Table: html_addElement(" + row + "," + col + ") called");
  }

  // Disabled, for now.
  public void html_add(Object o) throws HGLException {
    throw new HGLException("addElement must be used for adding table cells.");
  }

  // Overriding HglHTMLObject toString
  public String toString() {
    String out = new String();
    int r, c;
    TableCell cell;
    HashMap rh;

    out += openElem(openTag()) + "\n";

    // Outer loop: rows
    for (r=0; r<=highRow; r++) {

      out += openElem("<tr>") + "\n";

      // Output cells in this row, if any
      if ( (rh = (HashMap)tableRows.get(new Integer(r))) != null ) {
   for (c=0; c<=highCol; c++) {
     if ( (cell = (TableCell)rh.get(new Integer(c))) != null)
       out += cell.toString();
     else
       out += indent("<td>&nbsp;</td>") + "\n";
   }
      }
      else        // No cells here..just fill in row with empty cells
   for (c=0; c<=highCol; c++)
     out += indent("<td>&nbsp;</td>") + "\n";

      out += closeElem("</tr>") + "\n";
    }

    out += closeElem(closeTag()) + "\n";
    return out;
  } // public String toString()

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

} // class Table

--------------------
---- ./HGLTableCell.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLTableCell.java
// Purpose: A cell in an HGLTable.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;

class TableCell extends HglHTMLObject {

  private static String objType = "td";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);

  private static int cssCnt = 0;
  // Location of this cell in the table
  private int _row;
  private int _col;

  /* Constructor */
  public TableCell(Object o, int r, int c) {
   super(methods);
   objs.add(o);

   // Save this guy's location in the table
   _row = r;
   _col = c;
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }
  public int getRow() { return _row; }
  public int getCol() { return _row; }

} // class TableCell


--------------------
---- ./HGLUList.java
Author: Yan Koyfman
--------------------
//////////////////////////////////////////////////////////////////////
// Module: HGLUList.java
// Purpose: Unordered list HTML element.
//////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;

class UList extends HglHTMLObject {

  private static String objType = "ul";
  private static HglSymbolTable methods
    = new HglSymbolTable(null, false);

  private static int cssCnt = 0;

  /* Constructor */
  public UList() {
   super(methods);
  }

  public HglSymbolTable getWebMethods() { return methods; }
  protected String getObjType() { return objType; }
  protected int getCssCnt() { return cssCnt; }
  protected void incCssCnt() { cssCnt++; }

  public void html_addItem(Object o) throws HGLException {
   objs.add(new ListItem(o));
   HGLCommon.logger.finest("UList: html_addItem called");
  } // public void html_add(HglHTMLObject o)

  // Synonym for addItem
  public void html_add(Object o) throws HGLException {
   html_addItem(o);
   HGLCommon.logger.finest("UList: html_add called");
  } // public void html_add(HglHTMLObject o)

} // class UList


--------------------
---- ./HglHTMLObject.java
Author: Yan Koyfman
```

```
--------------------
////////////////////////////////////////////////////////////////////
// Module: HglHTMLObject.java
// Purpose: Base class for HTML objects.
////////////////////////////////////////////////////////////////////
import java.util.Vector;
import java.util.HashMap;
import java.util.Iterator;

abstract class HglHTMLObject {

  // Objects in this HTML object
  protected Vector objs = new Vector();
  protected static int indentLevel = 0;
  protected static String indentStr = "  ";
  protected String classRoot;
  protected String cssClassName;
  protected Boolean hasCssValues = new Boolean(false);
  protected Boolean hasPseudoClass = new Boolean(false);
  protected boolean hasNonCssValues = false;
  protected int cssCntCurrent;
  protected String tagOpen;
  protected String tagClose;
  protected static boolean nonCssProperty = true;
  protected boolean propLink = false;
  protected String propLinkStr;

  // Vector of available HTML classes
  public static String[] htmlClasses = {"Table", "Paragraph", "OList", "UList", "Link",
"Image", "Page", "Span"};

  // CSS classes that have been written
  protected static HashMap cssClasses = new HashMap();

  // CSS psedo classes that have been written
  protected static HashMap cssPseudoClasses = new HashMap();

  // CSS Properties that can be set on this object
  protected HashMap props = new HashMap();

  // Non-Css Properties that can be set on this object
  protected HashMap nonCssProps = new HashMap();

  // pseudo-classes - each one has its own properties hash!
  protected HashMap pseudoClasses = new HashMap();

  /* Constructor */
  public HglHTMLObject(HglSymbolTable methods) {
    Class c = getClass();
    String clsName = c.getName();
    HGLCommon.buildMethodTable(methods, c);
    HGLCommon.logger.finer("Constructing a " + clsName);
    initProps();
    classRoot = clsName;
    cssCntCurrent = getCssCnt();
    cssClassName = getObjType() + "." + classRoot + cssCntCurrent;
    incCssCnt();
  }

  // Properties common to all HTML objects
  private void initProps() {

    // CSS Properties
    props.put("font-size", new HGLProperty("font-size", getClass().getName(), "font-
size", false));
    props.put("font-style", new HGLProperty("font-style", getClass().getName(), "font-
style", false));
    props.put("text-align", new HGLProperty("text-align", getClass().getName(), "text-
align", false));
    props.put("font-family", new HGLProperty("font-family", getClass().getName(), "font-
family", false));
```

```
    props.put("font", new HGLProperty("font", getClass().getName(), "font", false));
    props.put("border", new HGLProperty("border", getClass().getName(), "border",
false));
    props.put("border-left", new HGLProperty("border-left", getClass().getName(),
"border-left", false));
    props.put("border-right", new HGLProperty("border-right", getClass().getName(),
"border-right", false));
    props.put("border-top", new HGLProperty("border-top", getClass().getName(), "border-
top", false));
    props.put("border-bottom", new HGLProperty("border-bottom", getClass().getName(),
"border-bottom", false));
    props.put("fgcolor", new HGLProperty("fgcolor", getClass().getName(), "color",
false));
    props.put("background", new HGLProperty("background", getClass().getName(),
"background", false));
    props.put("bgcolor", new HGLProperty("bgcolor", getClass().getName(), "background-
color", false));
    props.put("bgimage", new HGLProperty("bgimage", getClass().getName(), "background-
image", true));
    props.put("width", new HGLProperty("width", getClass().getName(), "width", false));
    props.put("height", new HGLProperty("height", getClass().getName(), "height",
false));
    props.put("float", new HGLProperty("float", getClass().getName(), "float", false));
    props.put("margin", new HGLProperty("margin", getClass().getName(), "margin",
false));
    props.put("margin-left", new HGLProperty("margin-left", getClass().getName(),
"margin-left", false));
    props.put("margin-right", new HGLProperty("margin-right", getClass().getName(),
"margin-right", false));
    props.put("margin-bottom", new HGLProperty("margin-bottom", getClass().getName(),
"margin-bottom", false));
    props.put("margin-top", new HGLProperty("margin-top", getClass().getName(), "margin-
top", false));
    props.put("padding", new HGLProperty("padding", getClass().getName(), "padding",
false));
    props.put("padding-bottom", new HGLProperty("padding-bottom", getClass().getName(),
"padding-bottom", false));
    props.put("padding-left", new HGLProperty("padding-left", getClass().getName(),
"padding-left", false));
    props.put("padding-right", new HGLProperty("padding-right", getClass().getName(),
"padding-right", false));
    props.put("padding-top", new HGLProperty("padding-top", getClass().getName(),
"padding-top", false));
    props.put("vertical-align", new HGLProperty("vertical-align", getClass().getName(),
"vertical-align", false));
    props.put("position", new HGLProperty("position", getClass().getName(), "position",
false));
    props.put("top", new HGLProperty("top", getClass().getName(), "top", false));
    props.put("bottom", new HGLProperty("bottom", getClass().getName(), "bottom",
false));
    props.put("left", new HGLProperty("left", getClass().getName(), "left", false));
    props.put("right", new HGLProperty("right", getClass().getName(), "right", false));
    props.put("line-height", new HGLProperty("line-height", getClass().getName(), "line-
height", false));
    props.put("z-index", new HGLProperty("z-index", getClass().getName(), "z-index",
false));
    // Template "user-name"  "user-name" "CSS-element-name"
    // props.put("", new HGLProperty("", getClass().getName(), "", false));

    pseudoClasses.put("focus", null);
    pseudoClasses.put("hover", null);
    pseudoClasses.put("active", null);

  } // private initProps()

  // Abstract get method for methods
  public abstract HglSymbolTable getWebMethods();

  // Get object type, needed for tags
  protected abstract String getObjType();
  protected abstract int getCssCnt();
  protected abstract void incCssCnt();

  // toString - how the HTML object creates its HTML code.
  public String toString() {
    String out = new String();

    out += openElem(openTag()) + "\n";

    for (int i=0; i<objs.size(); i++) {
      if ( !(objs.get(i) instanceof HglHTMLObject) )
   out += indent(objs.get(i).toString()) + "\n";
      else
   out += objs.get(i).toString() + "\n";
    } // for (int i=0; i<objs.size(); i++)

    if (hasCloseTag())
      out += closeElem(closeTag()) + "\n";
    else
      closeElem("");

    return out;
  } // public String toString()

  // Open and close tag with indentation level
  protected String openElem(String tag) {
    String s = new String();
    s += indent(tag);
    indentLevel++;
    return s;
  } // protected void openElem(String tag)

  // Open and close tag with indentation level
  protected String closeElem(String tag) {
    String s = new String();
    indentLevel--;
    s += indent(tag);
    return s;
  } // protected void openElem(String tag)

  // Output a string with the appropriate level of indentation
  protected String indent(String s) {
    String r = new String();
    for (int i=0; i<indentLevel; i++)
      r += indentStr;
    r += s;
    return r;
  } // protected void display(String s)

  // Output a string with the appropriate level of indentation
  protected String indent() {
    String r = new String();
    for (int i=0; i<indentLevel; i++)
      r += indentStr;
    return r;
  } // protected void display(String s)

  public void html_newline(Integer I) throws HGLException {
    for (int i=0; i<I.intValue(); i++)
      html_add("<br>");
  }

  public void html_add(Object o) throws HGLException {
    if (o instanceof Page)
      throw new HGLException("Cannot add Page objects");
    if (o instanceof String) {
      o = ((String)o).replaceAll("\\\n", "<br>");
    } // if (o instanceof String)
    objs.add(o);
    HGLCommon.logger.finest("html_add called");
  } // public void html_add(HglHTMLObject o)
```

```
  // Set a property on this element
  public void html_set(String name, Object o) throws HGLPropertyException {

    HGLProperty p;
    // Verify that this property is valid for this class
    if ( ((p = (HGLProperty)props.get(name)) == null) &&
    ((p = (HGLProperty)nonCssProps.get(name)) == null)) {
      throw new HGLPropertyException(name);
    } // if ( (Object o = props.get(name)) == null )
    p.setValue(o);

    // Check if this was a CSS property or non-css property
    if ( (HGLProperty)props.get(name) != null )
      hasCssValues = Boolean.TRUE;
    else
      hasNonCssValues = true;

  } // public void html_set(String name, Object o) throws HGLPropertyException

  // Set a pseudo-class property on this element
  public void html_setPseudo(String pc, String name, Object o) throws
HGLPropertyException {

    HGLProperty p;
    HashMap pcProps = (HashMap)pseudoClasses.get(pc);
    String propName;

    // Verify this is a valid pseudo-class
    if ( !pseudoClasses.containsKey(pc) )
      throw new HGLPropertyException(pc);
    else {

      // Check if a properties hash map is already associated with this
      // class
      if (pcProps == null) {
   // Create new properties hash map by duplicating the current one
   // and clearing any of its values
   pcProps = new HashMap();
   Iterator iProps = props.keySet().iterator();
   while (iProps.hasNext()) {
     propName = (String)iProps.next();
     pcProps.put(propName, new HGLProperty((HGLProperty)props.get(propName), true));
   } // while (iProps.hasNext())

   // Shiny new property hash for this pseudo-class
   pseudoClasses.put(pc, pcProps);
      } // if (pcProps == null)
    } // else

    // At this point we have a valid pc:(prop:val) structure
    // Set the value
    if ( ((p = (HGLProperty)pcProps.get(name)) == null) )
      throw new HGLPropertyException(name);
    else
      p.setValue(o);

    hasPseudoClass = Boolean.TRUE;
  } // public void html_setPseudo(String name, Object o) throws HGLPropertyException

  // Copy applicable properties from one object to another
  // Does not do any linking, merely for convience.
  public void html_inherit(HglHTMLObject other) {
    Iterator myProps = props.keySet().iterator();
    Iterator myPseudoClasses;
    HashMap otherPcMap, myPcMap;

    String key;
    HGLProperty p;

    // Try to set all applicable properties
    while (myProps.hasNext()) {
```

```
    key = (String)myProps.next();
    if ( (p = (HGLProperty)other.props.get(key)) != null )
 props.put(key, new HGLProperty(p, false));
  }

  // Copy pseudo-class properties
  myPseudoClasses = pseudoClasses.keySet().iterator();

  while (myPseudoClasses.hasNext()) {
    key = (String)myPseudoClasses.next();
    if ( (otherPcMap = (HashMap)other.pseudoClasses.get(key)) != null ) {

 // pcProps is a properties hash map for pseudo-class key;
 // now, for every property in this class, we create a new hash
 // table for this pseudo-class in this object
 myPcMap = new HashMap();
 pseudoClasses.put(key, myPcMap);
 myProps = props.keySet().iterator();
 while (myProps.hasNext()) {
   key = (String)myProps.next();
   if ( (p = (HGLProperty)otherPcMap.get(key)) != null )
     myPcMap.put(key, new HGLProperty(p, false));
 } // while (myProps.hasNext())
    } // if ( (otherPcMap = (HashMap)other.pseudoClasses.get(key)) != null )
  } // while (myPseudoClasses.hasNext())

  hasPseudoClass = other.hasPseudoClass;
  hasCssValues = other.hasCssValues;
}

// Create truly linked objects
public void html_link(HglHTMLObject other) {
  props = other.props;
  pseudoClasses = other.pseudoClasses;
  propLink = true;
  propLinkStr = other.getCssClass();
  hasCssValues = other.hasCssValues;
  hasPseudoClass = other.hasPseudoClass;
}

// The properties this object needs to define its appearence.
// Should be placed in the style sheet when printing file.
public String style() {
  String ret = "";
  Object o;
  HGLProperty p;
  Iterator iProps, cProps;
  HashMap hProps;
  String pcName, pcCssName;

  // Objects' styles
  for (int i=0; i<objs.size(); i++) {
    o = objs.get(i);
    if (o instanceof HglHTMLObject)
 ret += ((HglHTMLObject)o).style();
  }

  /*-----------------------------------------------------------------
   * CSS Classes
   */
  iProps = props.values().iterator();
  if ( !(cssClasses.containsKey(cssClassName)) && hasCssValues.booleanValue() &&
  iProps.hasNext() && !propLink ) {
    cssClasses.put(cssClassName, Boolean.TRUE);

    // Start class definition
    ret += cssClassName + "\n{";

    // Have each property output itself
    while (iProps.hasNext()) {
 p = (HGLProperty)iProps.next();
```

```
   if ( (p != null) && p.isSet() )
     ret += "\n  " + p.toString();
      }
      // Close class definition
      ret += "\n}\n";
    } // !iProps.hasNext() || propLink) )

    /*------------------------------------------------------------------
     * Pseudo-classes
     */
    if (hasPseudoClass.booleanValue() && !propLink) {

      cProps = pseudoClasses.keySet().iterator();

      // Have each property output itself
      while (cProps.hasNext()) {
   pcName = (String)cProps.next();
   hProps = (HashMap)pseudoClasses.get(pcName);

   if (hProps != null) {      // This pseudo-class has properties
     pcCssName = cssClassName+":"+pcName;

     if (!(cssPseudoClasses.containsKey(pcCssName))) {
       cssPseudoClasses.put(pcCssName, Boolean.TRUE);
       ret += pcCssName + "\n{";
       iProps = hProps.values().iterator();
       while (iProps.hasNext()) {
         p = (HGLProperty)iProps.next();
         if ( (p != null) && p.isSet() )
      ret += "\n  " + p.toString();
       }
       // Close class definition
       ret += "\n}\n";
     } // if (!(cssPseudoClasses.containsKey(cssClassName+":"+pcName)))
   } // if (hProps)
      } // while (cProps.hasNext())
    } // if (hasPseudoClass.booleanValue())

    return ret;
  }

  // The opening / closing code for a tag
  public String openTag() {
    String ret = "<" + getObjType();
    HGLProperty p;

    if (hasCssValues.booleanValue())
      if ( propLink )
   ret += " class=\"" + propLinkStr + "\"";
      else
   ret += " class=\"" + classRoot + cssCntCurrent + "\"";

    if (hasNonCssValues) {
      Iterator iProps = nonCssProps.values().iterator();
      // Have each property output itself
      while (iProps.hasNext()) {
   p = (HGLProperty)iProps.next();
   if ( (p != null) && p.isSet() )
     ret += " " + p.toString();
      }
    }
    return ret + ">";
  }

  public String closeTag() { return "</" + getObjType() + ">"; }
  public boolean hasCloseTag() { return true; } // Override if no close tag, eg <img>
  public String getCssClass() { return classRoot + cssCntCurrent; }
} // class HglHTMLObject
--------------------
---- ./HglSymbolTable.java
Author: Tim Kaczynski, Yan Koyfman
```

```
--------------------
///////////////////////////////////////////////////////////////////////
// Module: HglSymbolTable.java
// Purpose: Symbol table for variables and functions
///////////////////////////////////////////////////////////////////////

public class HglSymbolTable extends java.util.HashMap
{
  private HglSymbolTable _parent;
  private boolean _isFcn;
  public HglSymbolTable(HglSymbolTable parent, boolean isFcn)
  { super(); _parent = parent; _isFcn = isFcn; }
  /*----------------------------------------------------------*/
  /* Implementation of containsKey assumes that variables can  */
  /* not be re-declared in a nested scope unless that nested   */
  /* scope is a function.                                      */
  /*----------------------------------------------------------*/
  public boolean containsKey(Object o)
  {
    if (super.containsKey(o)) return true;
    else return false;
  }
  public Object get(Object key)
  {
    Object val = super.get(key);
    if ((val == null) && (isParentSearchable()))
      val = _parent.get(key);
    return val;
  }
  public HglSymbolTable getParent() { return _parent; }
  private boolean isParentSearchable()
  { return ((_parent != null) && (!_isFcn)); }
}


--------------------
---- ./HglVariable.java
Author: Tim Kaczynski, Yan Koyfman
--------------------
///////////////////////////////////////////////////////////////////////
// Module: HglVariable.java
// Purpose: The 'value' object in a symbol table; holds value of
// variable and code of functions.
///////////////////////////////////////////////////////////////////////

import antlr.collections.AST;

// Can have two meanings:
// 1) Variable declarations in _currentSymbolTable
// 2) Function declarations/definitions in _functionSymbolTable
public class HglVariable
{
  // return type in functions
  private int _type;
  // Value or symbol table for arguments in functions
  private Object _value;
  // Null until fundef seen
  private Object _DefAST;
  // Array size
  private int _arraySize = 0;
  private boolean _isArray = false;
  // constructor
  public HglVariable(int type)
  {
   _type = type;
   _value = null;
   _DefAST = null;
  }
  // constructor for arrays
  public HglVariable(int type, int arraySize)
  {
   _type = type;
   _value = null;
   _DefAST = null;
   _arraySize = arraySize;
    _isArray = true;
  }

  public int getType() { return _type; }
  public Object getValue() { return _value; }
  public Object setValue(Object val) {_value = val; return val;}
  public Object setDefAST(AST val) {_DefAST = val; return val;}
  public Object getDefAST() {return _DefAST;}
  public boolean isDefSet() {return _DefAST != null;}
  public boolean isArray() { return _isArray; }
}
--------------------
---- ./TypeChecker.java
Author: Ed Mezarina, Tim Kaczynski, Yan Koyfman
--------------------
/**
 * Validates data types.
 */
public class TypeChecker implements HGLLexerTokenTypes
{
    /**
     * Constructor.
     */
    public TypeChecker()
    {
        /* Nothig interesting to do here right now.*/
    }

    /**
     * Validates that the input object is of the correct target type.
     * @param target The target type (integer constant).
     * @param value The value attempting to be assigned to the target
     *        type.
     * @return true if the assignment is valid, false if not.
     */
    public boolean validateType(int target, Object value)
    {
        HGLCommon.logger.finest("validateType entry");

        boolean valid = false;

        switch (target)
        {
            case (INTEGER):
            {
                if (value instanceof Integer) valid = true;
                break;
            }
            case (FLOAT):
            {
                if ((value instanceof Double) || (value instanceof Integer))
                    valid = true;
                break;
            }
            case (CHAR):
            {
                if (value instanceof Character) valid = true;
                break;
            }
            case (BOOLEAN):
            {
                if (value instanceof Boolean) valid = true;
                break;
            }
            case (STR):
            {
                if (value instanceof String) valid = true;
                break;
            }
            case (PARAGRAPH):
            {
                HGLCommon.logger.finest("Paragraph assignment ");
                if (value instanceof Paragraph) valid = true;
                break;
            }
            case (OLIST):
            {
                HGLCommon.logger.finest("Ordered List assignment ");
                valid = false; // TODO: Remove when supported
                break;
            }
            case (ULIST):
            {
                HGLCommon.logger.finest("Unordered List assignment ");
                if (value instanceof UList) valid = true;
                break;
            }
            case (PAGE):
            {
                HGLCommon.logger.finest("Page assignment ");
                if (value instanceof Page) valid = true;
                break;
            }
            case (TABLE):
            {
                HGLCommon.logger.finest("Table assignment ");
                if (value instanceof Table) valid = true;
                break;
            }
            case (IMAGE):
            {
                HGLCommon.logger.finest("Image assignment ");
                if (value instanceof Image) valid = true;
                break;
            }
            case (LINK):
            {
                HGLCommon.logger.finest("Link assignment ");
                if (value instanceof Link) valid = true;
                break;
            }
            case (SPAN):
            {
                HGLCommon.logger.finest("Span assignment ");
                if (value instanceof Span) valid = true;
                break;
            }
        }

        HGLCommon.logger.finest("validateType exit: " + valid);

        return valid;
    }

    /**
     * Validates the data provided according to HGL grammar rules.
     * @param a Input one.
     * @param b Input two.
     * @return Object type to be returned to caller.
     */
    public Object validateData(Object a, Object b, int op)
    {
        boolean valid = true;
        String m = "Invalid Parameter type for this operation";
        Object r = null;

        //------------------------------------------------------------
        // For arithmetic operations we will only accept
        // integer and float numbers.
        //------------------------------------------------------------
        if (op == PLUS || op == MINUS ||
            op == TIMES || op == DIV || op == UMINUS || op == UPLUS)
        {
     if (!((a instanceof Number) && (b instanceof Number)) && ((op != UMINUS)&&(op !=
UPLUS)))
            {
                throw new RuntimeException(m);
            }
       if ( ( (op == UMINUS) || (op == UPLUS) ) && ( (!(a instanceof Number)) || (b !=
null)))
            {
         throw new RuntimeException("Unary negation / addition: " + m);
            }

            else
            {
                Number an = (Number)a;
                Number bn = (Number)b;
                boolean isFloat = ((a instanceof Double) ||
                                   (b instanceof Double));

                if (op == PLUS)
                {
                    if (isFloat)
                    {
                        r = new Double(an.doubleValue() + bn.doubleValue());
                    }
                    else
                    {
                        r = new Integer(an.intValue() + bn.intValue());
                    }
                }
                else if (op == MINUS)
                {
        if (isFloat)
                    {
            r = new Double(an.doubleValue() - bn.doubleValue());
                    }
        else
                    {
            r = new Integer(an.intValue() - bn.intValue());
                    }
                }
                else if (op == UMINUS)
        {
                    if (isFloat)
            {
                        r = new Double(-an.doubleValue());
            }
                    else
            {
                        r = new Integer(-an.intValue());
            }
        }
                else if (op == UPLUS)
        {
                    if (isFloat)
            {
                        r = new Double(an.doubleValue());
            }
                    else
            {
                        r = new Integer(an.intValue());
            }
        }
                else if (op == TIMES)
        {
                    if (isFloat)
                    {
```

```
                r = new Double(an.doubleValue() * bn.doubleValue());
            }
            else
            {
                r = new Integer(an.intValue() * bn.intValue());
            }
        }
        else if (op == DIV)
        {
            if (isFloat)
            {
                r = new Double(an.doubleValue() / bn.doubleValue());
            }
            else
            {
                r = new Integer(an.intValue() / bn.intValue());
            }
        }
    }
}
//--------------------------------------------------------------
// For conditional operations we will only accept
// boolean values.
//--------------------------------------------------------------
else if ((op == COND_AND) || (op == COND_OR))
{
    if (!(a instanceof Boolean) ||
        !(b instanceof Boolean))
    {
        throw new RuntimeException(m);
    }
    else
    {
        if (op == COND_OR)
        {
            r = (((Boolean)a).booleanValue() || ((Boolean)b).booleanValue()) ?
                new Boolean(true):new Boolean(false);
        }
        else
        {
            r = (((Boolean)a).booleanValue() && ((Boolean)b).booleanValue())?
                new Boolean(true):new Boolean (false);
        }
    }
}
//--------------------------------------------------------------
// String concatenation
//--------------------------------------------------------------
else if (op == STR_CAT)
{
r = new String(((String)a.toString()) + ((String)b.toString()));
}
//--------------------------------------------------------------
// For relational operations we will only accept
// integer and float numbers.
//--------------------------------------------------------------
else if (op == REL_EQ || op == REL_NEQ  ||
         op == REL_GT  || op == REL_GTE  ||
         op == REL_LT  || op == REL_LTE)
{
    if (!((a instanceof Number) && (b instanceof Number)))
    {
        throw new RuntimeException(m);
    }
    else
    {
        Number an = (Number)a;
        Number bn = (Number)b;
        boolean isFloat = ((a instanceof Double) ||
                           (b instanceof Double));
        if (op == REL_EQ)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() == bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() == bn.intValue());
            }
        }
        else if (op == REL_NEQ)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() != bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() != bn.intValue());
            }
        }
        else if (op == REL_GT)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() > bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() > bn.intValue());
            }
        }
        else if (op == REL_GTE)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() >= bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() >= bn.intValue());
            }
        }
        else if (op == REL_LT)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() < bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() < bn.intValue());
            }
        }
        else if (op == REL_LTE)
        {
            if (isFloat)
            {
                r = new Boolean(an.doubleValue() <= bn.doubleValue());
            }
            else
            {
                r = new Boolean(an.intValue() <= bn.intValue());
            }
        }
    }
}
//--------------------------------------------------------------
// If we are here, it means that the operation specified by
// the user is not supported by HGL. Report the error.
```

```
        //---------------------------------------------------------
        else
        {
            m = "Invalid Operation: \""+op+ "\"";
            throw new RuntimeException(m);
        }

        return r;
    }
}
--------------------
---- ./hgl.java
Author: Yan Koyfman, Tim Kaczynski, Shruti Gandhi
--------------------
import java.io.*;
import java.util.logging.*;
import java.util.regex.*;
import java.util.HashMap;
import java.util.Iterator;
import antlr.collections.*;

class hgl {

  public static PrintStream pOut = System.out;

  public static void main(String[] args) {
    try {

      // Simple command line options system
      HashMap optHash = new HashMap(); // opt regex:value
      int optind;                // first non-option argument
      boolean found;

      // Logging options: -d[cf] = message, class, function; cumulative
      String debugArgPattern = new String("d.{0,2}");
      optHash.put(debugArgPattern, Boolean.FALSE);

      // Tree display
      String displayTree = new String("t");
      optHash.put(displayTree, Boolean.FALSE);

      // Property display
      String getProps = new String("p");
      optHash.put(getProps, Boolean.FALSE);

      // Property display
      String printFile = new String("o.+");
      optHash.put(printFile, Boolean.FALSE);

      // Once all options are added, fetch an iterator.
      Iterator optStrings;
      String optString;

      // Check what command line arguments we've been passed.
      for (optind=0; optind<args.length; optind++) {

   // End of options?
   if ( !args[optind].startsWith("-") )
     break;

   // Reset iterator into hashmap
   optStrings = optHash.keySet().iterator();

   // Strip leading dash
   args[optind] = args[optind].substring(1);

   // Match any options defined in hash map? Note:
   // we do not verify correctness of suboptions!
   // If suboptions are present, they get set as the value,
   // otherwise the value is a Boolean true.
   found = false;
   while (optStrings.hasNext()) {
     optString = (String)(optStrings.next());
     if (args[optind].matches(optString)) {

       if (args[optind].length()>1)
         optHash.put(optString,args[optind].substring(1));
       else
         optHash.put(optString,Boolean.TRUE);

       found = true;
     } // if (Pattern.matches(optStrings[i], args[optind]))
   } // while (optStrings.hasNext())

   if (!found)
     throw new RuntimeException("Error: Invalid argument: " + args[optind]);
      } // for (int optind=0; optind<args.length; optind++)

      // Verify that a file has been provided
      if (args.length == 0) {
   System.err.println("usage: hgl -p | filename");
   System.exit(0);
      }

      // Check if user just wants property list
      if ( (optHash.get(getProps) instanceof Boolean) &&
      ((Boolean)optHash.get(getProps) == Boolean.TRUE) ) {

   String clsName;
   int i;

   // Initialize HTML classes
   for (i=0; i<HglHTMLObject.htmlClasses.length; i++) {
   // Create an instance of each of these classes to initialize properties vector
     clsName = (String)HglHTMLObject.htmlClasses[i];
     Class.forName(clsName).newInstance();
   }

   Iterator iProps = HGLProperty.getCssProps().iterator();
   while (iProps.hasNext())
     System.out.println((String)iProps.next());

   iProps = HGLProperty.getNonCssProps().iterator();
   while (iProps.hasNext())
     System.out.println((String)iProps.next());

   System.exit(0);
      }

      // Create main logger object
      Logger logger = Logger.getLogger("hglLog");

      // Create file and console handlers
      FileHandler fhandler = new
   FileHandler("log",   // name
          1024*64,   // max size
          2,      // max files
          true);  // append

      ConsoleHandler chandler = new ConsoleHandler();

      // Remove handlers, if any..we'll add our own
      Handler h[] = logger.getHandlers();
      for (int i=0; i<h.length; i++)
   logger.removeHandler(h[i]);

      // Set initial level
      //  The levels in descending order are:

      //     * SEVERE (highest value)
      //     * WARNING
      //     * INFO
      //     * CONFIG
      //     * FINE
      //     * FINER
      //     * FINEST (lowest value)
      chandler.setLevel(Level.FINEST); // Console handler
      fhandler.setLevel(Level.FINEST); // File handler
      logger.setLevel(Level.INFO); // Global value overrides handlers'

      // HGLLogFormatter cooks raw format string to our liking
      fhandler.setFormatter(new HGLLogFormatter(optHash.get(debugArgPattern)));
      chandler.setFormatter(new HGLLogFormatter(optHash.get(debugArgPattern)));

      // Add our handlers
      logger.addHandler(fhandler);
      logger.addHandler(chandler);

      // Verify optind is valid
      if (optind >= args.length) {
   System.err.println("usage: hgl -p | filename");
   System.exit(0);
      }

      // Save filename
      HGLCommon.currentFileName = args[optind];

     // Set up output stream
     String fname = null;
     if ((optHash.get(printFile) instanceof String) &&
        ((fname = (String)optHash.get(printFile)) != null)) {

      FileOutputStream out; // declare a file output object
      try
        {
         // Create a new file output stream
         // connected to "myfile.txt"
         out = new FileOutputStream(fname);

         // Connect print stream to the output stream
         hgl.pOut = new PrintStream( out );
        }
      catch (Exception e)
        {
         System.err.println ("Error writing to file " + fname);
        }
     } // ((Boolean)optHash.get(printFile) != null))

      HGLLexer lexer = new HGLLexer(new FileInputStream(args[optind]));
      HGLParser parser = new HGLParser(lexer);

      // Call to use the overridden class
      parser.setASTNodeClass("ASTLines");

      parser.start();

      if ( (optHash.get(displayTree) instanceof Boolean) &&
      ((Boolean)optHash.get(displayTree) == Boolean.TRUE) ) {
   antlr.debug.misc.ASTFrame frame =
     new antlr.debug.misc.ASTFrame("AST JTree Example", parser.getAST());
   frame.setVisible(true);
      } // ((Boolean)optHash.get(displayTree) == Boolean.TRUE) )
      HGLTreeWalker walker = new HGLTreeWalker();
      AST tree;

      if ( (tree = parser.getAST()) == null )
   System.exit(1);

      walker.start(tree); // First pass
      walker.start(tree); // Second pass
    }
    catch (HGLExceptionLine hel) {
      HGLCommon.scold(hel);
```

```
    } // } catch (HGLException he)
    catch (HGLException he) {
      System.err.println(HGLCommon.currentFileName+": Error: "+he.getMessage());
      //      he.printStackTrace();
    } // } catch (HGLException he)
    catch (antlr.RecognitionException re) {
      HGLCommon.scold(re);
      //      System.err.println("Error: " + HGLCommon.currentFileName
+":"+re.getLine()+": "+re.getMessage());
    }
    catch (Exception e) {
      System.err.println("Error: "+e);
      //      e.printStackTrace();
    }
  }
}
```