HGL: Language Reference Manual

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

October 20, 2005

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

Contents

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.

2 Lexical Conventions

2.1 Comments

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

2.2 Whitespace

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

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.

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
char            paragraph
const           olist
else            ulist
if              page
int             true
string          false
continue        break
return          void
```

`table` `image`

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 `""`.

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)

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

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

`,` Separate elements in lists
`;` Statement terminator
`$` When followed by a keyword, denote variable names

2.8 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.

2.9 Types

2.9.1 Available Types

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

Type	*Declaration Syntax*	*Sample Values*
String	`string s;`	`"My html page1";`
Integer	`int a;`	`-2147483648 to 2147483647`
Float	`float f;`	`3.4e+05 (see below)`
Boolean	`boolean b;`	`true` or `false`
Page	`page p;`	opaque type
Paragraph	`paragraph pg;`	opaque type
Table	`table t;`	opaque type
Image	`image i;`	opaque type
Unordered List	`ulist u;`	opaque type
Ordered List	`olist o;`	opaque type
Array	`ulist lists[5];`	[*positive integer*]

*A floating point number consists of an integer part, a decimal point, a fraction part, ('e'|'E'), 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.

2.9.2 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 *LRM Library* section.

2.10 Expressions

2.10.1 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.

2.10.2 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.

2.10.3 Function Calls

Function call syntax is similar to C. Examples:

```
print($webPage);              void functions not returning values
$var = fun();                 functions returning values
do_work(10, "a string");      example of argument list
```

2.10.4 Arithmetic Expressions

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

2.10.5 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.

2.10.6 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

2.10.7 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
`!`	`!v`	Returns `true` if v is false

2.10.8 Operator Precedence

Precedence rules were derived from PHP rules.

Operators	**Associativity**
`* /`	left *(highest)*
`+ -`	left
`< <= > >=`	non-associative
`== !=`	non-associative
`&&`	left
`||`	left
`=`	right *(lowest)*

2.11 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.

2.11.1 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. Examples:

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

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 one of two scopes, global or function-level. Variables declared outside of any function are visible globally and in all functions; variables defined within a function are only visible in that function.

2.11.2 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 HGL Library

3.1 Built-in HTML Objects

The following HTML constructs are represented as first-order primitives in the language.

- page `page`
  - **Functions**
  - `print()` - output HTML/CSS code for page object
  - `addElement(HTML object)` - add an HTML object to the page
  - **Attributes:type**
  - `bgcolor:int` - RBG value for bg color
- paragraph `paragraph`
  - **Functions**
  - **Attributes:type**
  - `bgcolor:int` - RBG value for bg color
  - `textType:string` - font face style
- unordered list `ulist`
  - **Functions**
  - `addItem(`*`item`*`)` - add list item; numeric or string
  - **Attributes:type**
  - `bgcolor:int` - RBG value for bg color
  - `fgcolor:int` - RBG value for text
- ordered list `olist`
  - **Functions**

  - `addItem(item)` - add list item; numeric or string
  - **Attributes:type**
  - `bgcolor:int` - RBG value for bg color
  - `fgcolor:int` - RBG value for text

- table `table`
  - **Functions**
  - `addElement(item, loc)` - add item to cell
  - **Attributes:type**
  - `bgcolor:int` - RBG value for bg color
  - `rows:int` - number of rows
  - `columns:int` - number of columns

- image `image`
  - **Functions**
  - `setPath(addr)` - path to image
  - **Attributes:type**
  - `border:boolean` - set border around image