AWE

Autonomous Web Explorer

Khai Vuong, George Sirois

December 20, 2005

Table of Contents

1 Introduction: White paper
2 Language Tutorial
3 Language Reference Manual
4 Project Plan
5 Architectural Design
6 Test Plan
7 Lessons Learned

Appendix A: Code Listing

1 Introduction: White paper

Language Overview

As we settle in to the information age, internet access is becoming more and more a basic commodity. In turn, the volume of information on the internet grows by enormous amounts every day. Google's page index is in the tens of billions of pages and increasing every day. Even with powerful search tools, the task of finding new and relevant information is daunting.

In light of this, we decided to implement a simple web-surfing language to help automate the web-browsing experience. The primary mission for the language is to allow the user to write a quick script to go to a web page and follow any links or download any files. With some basic logic and regular expression functionality, the user should be able to easily create scripts to tailor content delivery to his specific needs, without having to spend the time of manually tracking down the desired information.

2 Language Tutorial

Syntactically, we have decided to keep our language as much like C as possible, because the syntax of C is well known and has proven to be adequate. Modularity will be achieved via functions, and curly brackets ({ }) will allow the programmer to group together multiple statements within the same scope. The ; symbol will mark the end of a statement.

The primitive data types will include the int and string types. These are identical to their counterparts in most other languages. In addition, our language will also introduce the hyperlink data type, which will contain a url and whatever text that the link contains. The webfile type will store a web resource that can be manipulated by the programmer via our language. We would also introduce a collection type, which would act as a group of hyperlink data types.

The operators implemented in our language will include the simple mathematical operators, i.e. +, -, *, /, and %. The = operator will be used for assignment. Whenever a webfile type is assigned, it is at that time that the web resource is downloaded in temporary storage (with all of the relative urls expanded to absolute urls). The ==, !=, <, <=, >, and >= operators will be used for comparison. The && and || operators will be used for the logical AND and OR, respectively. For link traversal, we will use C's pointer notation. For example, to point a webpage to a hyperlink, you could use the following statements: page1 = *link1. Likewise, the following statement would assign a link to a webpage: link1 = &page1;. Another operator that we would like to introduce would be the save operator (->). This operator would be applied from a webfile to some local file location. If the specified local location is a directory, -> will save to the default filename, which is the url of the webfile with all illegal characters replaced by underscore characters. For example, the statement *{http://www.yahoo.com/index.htm, “”} -> "C:\\downloaded"; would save to the C:\downloaded\http___www.yahoo.com_index.htm the contents of page1. The indexer [] will be used to access members of collection. Member can be indexed via the position [int] or the text of the link [string]. Finally, we would also like to include the ~~ and !~ operators for comparisons between strings and regular expressions. The statement (a ~~ b), where a is a string and b is a regular expression would return 1 if b matches a and 0 if b does not match a.

Our conditional statements will be limited to the if(...)...else... statement.

Looping structures will include the do while(…)… loop and a foreach(… in …)… loop.

Our language would also include some built-in functions to aid the programmer:

- url(hyperlink) would return the url of the given hyperlink
- title(hyperlink) would return the text of the given hyperlink
- count(collection) would return the length of the given collection
- add(collection, hyperlink) would add a hyperlink to a collection
- remove(collection, hyperlink) would remove a hyperlink from a collection
- clear(collection) would remove all hyperlinks from a collection
- status(webfile) would return the status of the http request of the given file (200, 404, etc.)
- isSuccess(webfile) would return if the status of the given file is 200
- type(webfile) would return the MIME type of the webfile
- isHtml(webfile) would return if the type of the given file is text/html
- isImage(webfile) would return if the type of the given file is image/
- html(webfile) would return the html source code of the given webfile.
- links(webfile) would return a collection of all links on the given webpage.
  images(webfile) would return a collection of all images on the given webpage.
- print(string) prints the given string to the console
- save(string, filename) saves the given string to the specified file
- indexOf(string1, string2, start, length) returns the index of string2 in string1 in the given bounds
- length(string) returns the length of the given string
- replace(string1, string2, string3) returns a copy of string1 inwhich all instances of string2 have been replaced by string3
- delete(string, start, length) returns a copy of the given string without the given bounds
- subString(string, start, length) returns a substring from the given string within the given bounds

Of the above functions, isSuccess is built on top of status. isHtml and isImage are built on top of type. title, text, links, and images are built on top of html. replace and delete are built on top of indexOf and subString. They are just helper functions to easier the job of programmer.

A sample program written in the above described language follows:

```
void main()
{
  hyperlink currentLink;
  webfile currentPage;

  currentLink = {"http://www.google.com", ""};
  currentPage = *currentLink;

  while((isSuccess(currentPage)) && (isHtml(currentPage))
  {
    foreach(hyperlink image in images(currentPage))
    {
      *image -> "C:\\downloaded_files";
    }
    currentLink = link(currentPage, 0)
    currentPage = *currentLink;
  }

  print("Done!");
}
```

The main() function is just like you'd expect from C. A link (currentLink) and webfile (currentPage) are declared, and currentLink is pointed to Google's website. Then, while currentLink points to a web page, currentPage is assigned to that page, all of the images are saved to C:\downloaded_files, and then currentLink is pointed to the first link on currentPage.

Another, more complex, example would be as follows:

```
void main()
{
  hyperlink lnk = {"http://www.google.com", ""};

  // Visit google.com
  visit(lnk, "c:\\data", 0);

  print("Done!");
}

// The visit() func is a recursive function that saves a webpage and all
// of the images on it, if it matches the regular expression ".* Brin.*"
// Then, it recursively calls itself on every webpage that current webpage
// links to a depth of 3 from the original page, if the url of the link
// matches the regular expression ".*about.*" or the link containst he text // "About
Google".
void visit(hyperlink lnk, string dir, int depth)
{
```

```
  webfile page1 = *lnk;

  // Check if the page has been downloaded
  if(!isSuccess(page1) || !isHtml(page1))
  {
    print("Download error!");
  }
  else
  {
    // Check if page1 contains text that matched ".* Brin.*"
       if (text(page1) ~~ ".* Brin.*")
       {
         // Save the webpage
         page1 -> dir;

         // Save every image on the webpage
         collection images1 = images(page1);
         foreach (hyperlink img1 in images1)
         {
           *img1 -> dir;
         }
       }

       // Increment the depth
       depth++;

       if (depth < 3)
       {
              // Follow each link whose url matches ".*about.* "
              // or whose text is "About Google"
              collection links1 = links(page1);
              foreach (hyperlink link in links1)
              {
                if ((url(link1) ~~ ".*about.*") ||
                    (text(link1) == "About Google"))
                {
                  visit(link1, dir, depth);
                }
              }
    }
  }
}
```

3 Language Reference Manual

3.1 Introduction

The intent of Autonomous Web Explorer (AWE) is to allow users to create simple scripts in order to automate the browsing of web pages. The language is syntactically small and does not contain an abundance of library functions – only enough to get the very basics done. In this respect, the syntax is very much modeled after C. This allows the user to pick up the syntax very quickly yet still write powerful scripts that gather the information they need.

3.2 Structure

The structure of an AWE script is taken directly from C. The main loop of the program is denoted by:

```
void main()
{
	//CODE
}
```

Further, the programmer may declare functions in the C style with a return type, arguments inside of the parenthesis, and return types.

```
int sum(int i, int k)
{
	return i + k;
}
```

3.3 Lexical Conventions

3.3.1 Comments

Comments are in the traditional C++ style where a double slash '//' at the beginning of a line denotes the entire line as a comment.

3.3.2 Identifiers

A sequence of letters and numbers is considered an identifier. AWE retains C's case sensitivity. An identifier may not start with a number and may contain the underscore "_" character.

3.3.3 Keywords

There are a number of identifiers which are reserved as keywords and may not be used as user-defined identifiers:

int	else	return	bool
string	in	break	
hyperlink	while	if	

webfileforeach collection

3.3.4 Constants

3.3.4.1 String Constants

A string constant is a sequence of characters surrounded by a double quote. There is no char type so any constant must be surrounded by double quotes. The forward slash “\” is used to escape special characters such as newlines, carriage returns, etc. A double slash “\\” inserts a single slash into the string constant. A newline is given by “\n” and a carriage return is given by “\r”.

3.3.4.2 Integer Constants

An integer constant is a sequence of digits (0-9).

3.3.5 Separators

The curly brackets ({ }) denote the beginnings and endings of functions and allow the programmer to group code blocks in the same scope.

The semicolon “;” denotes the end of a statement. Multiple statements may be placed on a single line.

White space is ignored for anything other than the purpose of separating syntactic elements.

3.3.6 Operators

AWE contains a somewhat large set of operators. They are simply listed here and will be discussed in more detail in the operators section.

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

3.4 Types

AWE has two different types – value types and standard types.

3.4.1 Value types

Value types are integers and strings. They are essentially an area of allocated memory which stores a primitive type. Certain operators operate only on value types. Value types are:

int
string
bool

3.4.1.1 int

An integer is 32 bits allocated in memory to store integer values. Integers are signed.

3.4.1.2 string

A string is dynamically allocated and stores a sequence of Unicode characters.

3.4.1.3 bool

A bool stores a value of either true or false. It may be the result of comparison operations, and boolean logical operations may be performed on it.

3.4.2 Standard types

Standard types are the more complicated types built into the language. They may contain several information blocks and must be manipulated using the built in functions. Certain assignment operators may only be used on standard types. Standard types are:

hyperlink
webfile
collection

3.4.2.1 hyperlink

Contains the url of the link and the title of the page. A hyperlink is declared as follows:

```
hyperlink l;
l = { "http://www.google.com" , "abc"};
```

where http://www.google.com is the url and “abc” is the title.

All components of a hyperlink may be accessed by standard library functions.

3.4.2.2 webfile

Stores a resource pointed to by a link. Contains MIME type information and Status information. A webfile is declared as follows:

```
webfile w;
w = *l;
```

where l is a valid hyperlink.

3.4.2.3 collection

Stores a collection of hyperlinks. Individual hyperlinks be accessed using library functions.

3.5 Operator Definitions

Precedence and Associativity of Operators

Operators	Associativity
()	Left to right
++ -- * & - !	Right to left
* / %	Left to right
+ - ^	Left to right
< <= > >=	Left to right
== != ~~ !~	Left to right
&&	Left to right
||	Left to right
=	Right to left
->	Left to right
,	Left to right

3.5.1 Grouping and Indexing Operators

Grouping operators are used to modify the order of evaluation by specifying that everything within the operators is to be evaluated with the highest precedence.

3.5.1.1 (*expressions*)

Groups the contained expressions into a group of highest precedence

3.5.2 Unary Operators

Unary operators may only be used on integer types.

3.5.2.1 *int*++

Increments an integer value by 1.

3.5.2.2 *int*—

Decrements an integer value by 1.

3.5.2.3 –*int*

Invert the sign of an integer.

3.5.2.4 !bool

Takes the complement of a boolean

3.5.3 Multiplicative Operators

Multiplicative operators may only be used on integer types and result in integers.

3.5.3.1 (*integer expression*) * (*integer expression*)

Multiplies the two integer expressions.

3.5.3.2 (*integer expression*) / (*integer expression*)

Divides the two integer expressions. Returns only the quotient.

3.5.3.3 (*integer expression*) % (*integer expression*)

Takes the modulus of two integers. Returns the remainder.

3.5.4 Additive Operators

Additive operators may only be used on integer types and result in integers.

3.5.4.1 (*integer expression*) + (*integer expression*)

Adds the two integer expressions.

3.5.4.1 (*integer expression*) - (*integer expression*)

Subtracts the two integer expressions.

3.5.5 Assignment Operators

Assignment operators may be performed on all types and simply copy the value of one expression to another.

3.5.5.1 *variable = expression*

Sets the *variable* to the value of the expression provided they are of the same type.

3.5.5.2 *webfile = *hyperlink*

Similar to C's pointer notation, this points a webpage to a hyperlink.

3.5.5.3 *hyperlink = &webfile*

Similar to C's pointer notation, this points a hyperlink to a webpage.

3.5.5.4 *webfile -> string*

The assignment operator makes it easy to pipe the data contained in a *webfile* to a path on disk as specified by *string.*

As an example of the above 3 operators,
*{http://www.yahoo.com/index.htm, “”} -> “C:\\downloaded”; would save the index file to
C:\downloaded\http___www_yahoo_com_index_htm
where all non alphanumeric characters are replaced by underscores. In the event of name collisions or duplicate files, the original files are appended, not overwritten.

3.5.6 Comparison Operators

Most comparison operators may only be used on integer types. There are a limited number which may be used on string types.

3.5.6.1 (*integer expression*) == (*integer expression*)

Evaluates to true if both integers are equal. Evaluates to false otherwise.

3.5.6.2 (*integer expression*) != (*integer expression*)

Evaluates to false if both integers are equal. Evaluates to true otherwise.

3.5.6.3 (*integer expression a*) < (*integer expression b*)

Evaluates to true if a is less than b. Evaluates to false otherwise.

3.5.6.4 (*integer expression a*) <= (*integer expression b*)

Evaluates to true if a is less than or equal to b. Evaluates to false otherwise.

3.5.6.5 (*integer expression a*) > (*integer expression b*)

Evaluates to true if a is greater than b. Evaluates to false otherwise.

3.5.6.6 (*integer expression a*) >= (*integer expression b*)

Evaluates to true if a is greater than or equal to b. Evaluates to false otherwise.

3.5.6.7 (*boolean expression*) && (*boolean expression*)

Performs the logical AND of two boolean expressions. Does not short-circuit.

3.5.6.8 (*boolean expression*) || (*boolean expression*)

Performs the logical OR of two boolean expressions. Does not short-circuit.

3.5.6.9 (*string expression a*) ~~ (*string expression b*)

Returns true if expression a matches the REGEX given by the string in expression b, or false otherwise.

3.5.6.10 (*string expression a*) !~ (*string expression b*)

Returns false if expression a matches the REGEX given in expression b, or true otherwise.

3.5.6.11 (*string expression*) ^ (*string expression*)

Returns the concatenation of the two strings.

3.6 Flow Control

3.6.1 The *do while* statement

```
do
{
        //Execute this
}
while (condition is true)
```

The while loop evaluates the logical syntax of the condition in the parenthesis. If it evaluates to true, the code in the braces begins execution. When the end brace is reached and the condition is reevaluated. If there is only a single line of code to be executed, it may directly follow the while statement with no braces.

3.6.2 The *for* statement

```
for(variable = int; variable comparison int2; variable++ or variable--)
{
        //Execute
}
```

The for statement allows the programmer to assign an integer value to variable. Then, as long as "variable comparison *int2"* returns true, the loop executes. At the end of the loop, the variable is incremented or decremented and the evaluation is made again. The loop continues until the middle condition is no longer satisfied. While this is the behavior of the for loop, the actual syntax is (expr; expr; expr), and continues operation while the middle expr is satisfied.

3.6.3 The *foreach* statement

```
foreach(hyperlink x in collection)
{
        //Do something with x
}
```

The foreach statement allows the programmer to easily traverse a collection. For each item in the collection of type *type*, it is assigned to the local variable "name" and the loop is run.

3.6.4 The *if else* construct

```
if (condition 1)
{
        //statement 1
}
else if (condition 2)
{
        //statement 2
}
else
{
        //statement 3
}
```

The *if else* construct first evaluates condition 1 for truth. If it is true, statement 1 is executed. If not, it checks condition 2. If condition 2 is true, statement 2 is executed. Finally, if neither condition is met, statement 3 is executed. Note that the *else if* and *else* are optional, and the programmer may chain together as many *else if*s as he or she wishes.

3.6.5 The *return [argument]* statement

Calling *return* in a function immediately breaks execution and returns the value of *argument* to the point where the function was called. For void functions, return may be called with no argument.

3.6.6 The *break* statement

Calling *break* automatically jumps execution out of the lowest-level loop that the program is executing. Break is ignored outside of a loop body.

3.7 Declaration Syntax

3.7.1 Value type variables

Value type variables are declared as *type identifier*; or *type identifier* = *value*; Where value can be an expression which returns the correct value type. *Value* may also be a constant (*string* or *int*).

3.7.2 Standard type variables

Standard type variables are declared as *type identifier*; or *type identifier* = *value*; If they are assigned to a value, its type must match the identifier type.

While there are no constants for standard type variables, they can be created on the fly.

For example:

hyperlink link1 = *{“http://www.yahoo.com/index.htm”,””};

This is a link to Yahoo’s index file. The * denotes the creation of a link from a webpage, the first string is the link to the page, and the second string is the optional title the programmer may assign.

collection collection1 = {};

This is an empty collection. Collections may only contain hyperlinks. To add links, place them between the braces separated by commas.

3.8 Standard Library

The following functions are built in to the language:

3.8.1 *string* url (*hyperlink*)

Returns the url of the given hyperlink as a *string.*

3.8.2 *string* text (*hyperlink*)

Returns the text of the given hyperlink as a *string.*

3.8.3 *int* count (*collection*)

Returns the length of the given *collection* as an *int.*

3.8.4 *void* add(*collection*, *hyperlink*)

Adds a *hyperlink* to a *collection*.

3.8.5 *void* remove (*collection*, *hyperlink*)

Removes a *hyperlink* from a *collection*.

3.8.6 *void* clear(*collection)*

Removes all hyperlinks from a collection.

3.8.7 *int* status(*webfile)*

Returns the status of the http request of the given file (200, 404, etc.) as an *int.*

3.8.8 *int* isSuccess (*webfile)*

Returns *int* 1 if the status of the given file is 200. Returns *int* 0 otherwise.

3.8.9 *string* type (*webfile*)

Returns the MIME type of the webfile as a *string.*

3.8.10 *int* isHtml (*webfile)*

Returns *int* 1 if the type of the given file is text/html. Returns *int* 0 otherwise.

3.8.11 *int* isImage (*webfile)*

Returns *int* 1 if the type of the given file is image. Returns *int* 0 otherwise.

3.8.12 *string* html (*webfile*)

Returns the html source code of the given webfile as a *string.*

3.8.13 *collection* links (*webfile*)

Returns a collection of all the links on the given webfile as a *collection.*

3.8.14 *collection* images (*webfile*)

Returns a collection of all the links which lead to images on the given webfile as a *collection.*

3.8.15 *void* print (*string*)

Prints the given *string* to the console.

3.8.16 *void* save (*string, string* filename)

Saves the given *string* to path filename.

3.8.17 *int* indexOf (*string* string1, *string* string2, *int* start)

Returns the index of string2 in string1 in the given bounds as an *int*. Returns -1 if not found.

3.8.18 *int* length (*string*)

Returns the length of the given string as an *int*.

3.8.19 *string* replace (*string* string1, *string* string2, *string* string3)

Returns a copy of string1 in which all instances of string2 have been replaced by string3.

3.8.20 *string* subString (*string* string1, *int* start, *int* length)

Returns a substring from the given string within the given bounds.

3.9 Regular Expressions

3.9.1 Regular Expression Syntax

We have chosen a basic subset of Java's regular expression library. It includes:

direct comparison ("hello" will match "hello" in a string)
line position indicators (^signifies the beginning of a line and $ signifies the end)
wildcard matching (using the '.')
Kleene star matching ("x*" matches zero or more consecutive xs)
range matching (a-z matches all lowercase characters a-z)
special escape characters (\w matches all alphanumeric characters)

4 Project Plan

Planning Process

Because our initial team leader (Mick Langford) and Khai already had a good idea before the group started, there was not much planning in terms of conceptualization. The majority of the planning consisted of several meetings in which the overall scope of the language was hashed out and group roles were determined. After this point, the group became extremely fragmented and most of the planning was done by email between George and Khai with some help from the TA (Chris Conway).

Project Timeline

For the early part of the semester, the project was on a solid timeline. We met the deadlines for the language whitepaper and the LRM easily. However, once Mick dropped the class, all planning went out the window. The project was implemented in a very haphazard manner, with direct communication between George and Khai and some group meetings with the TA (Chris Conway).

Team Member Roles

Michael Langford (dropped)	Language design (part of)
Khai Vuong	Language design (part of), Architect Front-end: Lexer, Parser, TreeWalker Back-end: Interpreter, Symbol Resolver (primary)
George Sirois	Back-end: Interpreter, Symbol Resolver (assisting), Type system, Built-in function library Documentation

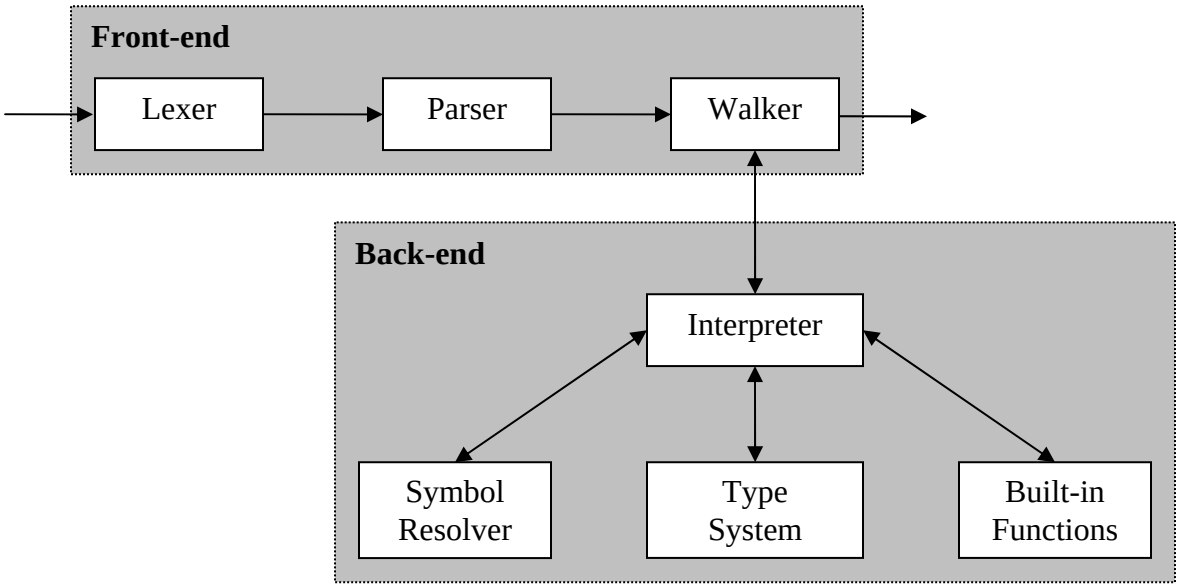
Development Environment

Development occurred exclusively in Eclipse running on windows. We hosted a private FTP server to use as a file dump, and George ran a CVS repository on his machine. Due to the turbulent nature of the group dynamic and the extreme isolation of each component until very late in the development process, this was rarely used.

5 Architectural Design

The AWE interpreter consist of

- A lexer that parses console input to tokens
- A parser that parses the tokens and builds the AST
- A tree walker that walks the AST and calls the interpreter
- An interpreter that does the execution (lite-version)
- A symbol resolver that tracks functions and variables (lite-version)
- A type system that provides support for basic (Boolean, Integer, and String) and complex (Hyperlink, Webfile, and Collection) types
- A set of built-in functions



The lexer, parser, and tree walker are implemented using ANTLR
The other components are implement using Java

6 Test Plan

Due to problems within the group, we had very little to test right up to the last minute. Because this was the case, each individual component was tested extensively by its author. For example, Khai tested the Lexer/Parser by running exhaustive simulations of any construct he could think that it would have to parse.

George's job was a bit easier, since he was writing library functions. Because they had specific tasks to do, they could be debugged in a very straightforward fashion: they either did what they were intended to do or they didn't.

Once we were able to assemble a "lite interpreter", we had so little time we were forced to debug on the fly. We tried to run cases which hit many features of the language, but were not overly complicated. A sample test script is as follows:

```
hyperlink l;
l = { "http://www.google.com" , "abc"};

webfile w;
w = *l;

if (isSuccess(w))
{
   print(status(w));
   print(type(w));
   print(html(w));
   collection c;
   c = (links(w));
   int i;
   i = count(c);
   print(i);

   if (i >= 2)
   {
        *getItemByIndex(c, 2) -> "C:\";
   }
   else
   {
        print("There are less than 3 links from " ^ url(l));
   }
}
else
{
   print("Failed");
}
$
```

7 Lessons Learned

Khai:

- This project is not hard as it seems to be
- Try to finish the work as soon as possible, because you will have no free time at the end of the semester
- Select your team members carefully

George:

- Slow and steady wins the race. Slowly nibbling away at the work will yield much better results than trying to do things in large hasty chunks.
- The TAs are very helpful, but you need to ask them questions in order to tap into their knowledge. They can't offer you useful, specific advice unless you ask for it.
- Try to get into a group where you already know some of the members.

Appendix A: Code Listing

```
/*
 * AweGrammar.g
 *
 * @author Khai Quang Vuong - kqv1@columbia.edu
 *
 */

class AweLexer extends Lexer;

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

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

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

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

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

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

LBRACE  :   '{';
RBRACE  :   '}';
LPAREN  :   '(';
RPAREN  :   ')';
LBRK    :   '[';
RBRK    :   ']';
MULT    :   '*';
PLUS    :   '+';
MINUS   :   '-';
DIV     :   '/';
MOD         :     '%';
CARET :     '^';
ASGN    :   '=';
INC         :     "++";
DEC         :     "--";
PLUSEQ  :   "+=";
MINUSEQ :   "-=";
MULTEQ  :   "*=";
DIVEQ   :   "/=";
MODEQ   :   "%=";
GT      :   '>';
LT      :   '<';
GE      :   ">=";
LE      :   "<=";
```

```
EQ      :   "==";
NEQ     :   "!=";
MT      :   "~~";
NMT     :   "!~";
COMMA   :   ',';
COLON   :   ':';
SEMI    :   ';';
EOP         :       '$';
NOT         :       '!';
SAVE  :     "->";
AMP         :       "&";
AND         :       "&&";
OR          :       "||";

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

INT         :       (DIGIT)+;

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

class AweParser extends Parser;

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

tokens {
//    PROGRAM;
      FUNCTION;
      PARAM_LIST;
      STATEMENT;
      DECLARE;
      EXPR_LIST;
      CALL;
      HYPERLINK;
      COLLECTION;
}


type        :       "bool"
                    |       "int"
                    |       "string"
                    |       "webfile"
                    |       "hyperlink"
                    |       "collection"
                    ;

link        :       LBRACE! STRING (COMMA! STRING)? RBRACE!
                          {#link = #([HYPERLINK,"HYPERLINK"], link); }
                    ;

collection  :       LBRK! link (COMMA! link)* RBRK!
                          {#collection = #([COLLECTION,"COLLECTION"],
collection); }
                    ;
```

```
program           :     ( ("void" | type ID LPAREN) => function |
statement )*
                        EOP!
                  {#program = #([STATEMENT,"PROG"], program); }
                  ;

function    :     (type | "void") ID LPAREN! param_list RPAREN!
                        LBRACE!
                              (statement)*
                        RBRACE!
                  {#function = #([FUNCTION,"FUNCTION"], function); }
                  ;

param_list  :     type ID ( COMMA! type ID )*
                        {#param_list = #([PARAM_LIST,"PARAM_LIST"],
param_list); }
            |   /* nothing */
                        {#param_list = #([PARAM_LIST,"PARAM_LIST"],
param_list); }
            ;

statement   :     decl_stmt
                  |     if_stmt
                    |   for_stmt
              | do_stmt
              | while_stmt
              | break_stmt
              | cont_stmt
                |     asgn_stmt
                |     call SEMI!
                  |   return_stmt
              | LBRACE! (statement)* RBRACE!
                  {#statement = #([STATEMENT,"STATEMENT"], statement); }
              ;

decl_stmt   :     type decl_item ( COMMA! decl_item )* SEMI!
                        {#decl_stmt = #([DECLARE,"DECLARE"], decl_stmt);
}
                        ;

decl_item   :     ID^ (ASGN! expression)?;

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

for_stmt    :     "foreach"^ LPAREN! "hyperlink"! ID "in"! ID RPAREN!
                        statement
            ;

do_stmt!    :     "do"^
                              statement
                  "while" LPAREN! expression RPAREN!
            ;

while_stmt! :     "while"^ LPAREN! expression RPAREN!
                              statement
                  ;

break_stmt  :     "break" SEMI!;

cont_stmt   :     "continue" SEMI!;
```

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

asgn_stmt   :     l_value
                        (
                              ( ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^ |
DIVEQ^ | MODEQ^ | SAVE^) expression
                              |     (INC^ | DEC^)
                        )
                        SEMI!;

call  :     ID LPAREN! expr_list RPAREN!
                  {#call = #([CALL, "CALL"], call); };

expr_list   :     expression ( COMMA! expression )*
                        {#expr_list = #([EXPR_LIST,"EXPR_LIST"],
expr_list); }
            |   /* nothing */
                  {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
            ;

index       :     INT | STRING;

l_value           :     ID;//^ ( LBRK! index RBRK! )?;

primitive   :     l_value
            |     call
            |     "true" | "false"
            |     INT
            |     STRING
            |     link
            |     collection
            |     LPAREN! expression RPAREN!
            ;

signed_expr :     (NOT^ | MINUS^ | PLUS^ | AMP^ | MULT^)? primitive;

mult_expr   :     signed_expr ( (MULT^ | DIV^ | MOD^) signed_expr )*;

add_expr    :     mult_expr ( (PLUS^ | MINUS^ | CARET^) mult_expr )*;

relat_expr  :     add_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^ | MT^ |
NMT^) add_expr )?;

and_term    :     relat_expr ( AND^ relat_expr )*;

expression  :     and_term ( OR^ and_term )*;
```

```
/*
 * AweWalker.g
 *
 * @author Khai Quang Vuong - kqv1@columbia.edu
 *
 */

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

class AweAntlrWalker extends TreeParser;
options {
    importVocab = AweAntlr;
}

{
    static AweDataType null_data = new AweDataType( "<NULL>" );
    AweInterpreter ipt = new AweInterpreter();
}

type        :     "bool"
                  |     "int"
                  |     "string"
                  |     "webfile"
                  |     "hyperlink"
                  |     "collection"
                  ;

expr returns [ AweDataType r ]
{
    AweDataType a, b;
    Vector v;
    AweDataType[] x;
    String s = null;
    String[] sx;
    r = null_data;
}
        : #(OR a=expr right_or:.)
            { r = ( ((AweBoolean)a).data ? a : expr(#right_or) ); }
        | #(AND a=expr right_and:.)
            { r = ( ((AweBoolean)a).data ? expr(#right_and) : a ); }
          | #(NOT a=expr)               { r = a.complementBoolean(); }
          | #(GE a=expr b=expr)         { r = a.greaterThanEqual( b ); }
          | #(LE a=expr b=expr)         { r = a.lessThanEqual( b ); }
          | #(GT a=expr b=expr)         { r = a.greaterThan( b ); }
          | #(LT a=expr b=expr)         { r = a.lessThan( b ); }
          | #(EQ a=expr b=expr)         { r = a.equal( b ); }
          | #(NEQ a=expr b=expr)        { r = a.notEqual( b ); }
          | #(PLUS a=expr b=expr)       { r = a.plus( b ); }
          | #(MINUS a=expr b=expr)      { r = a.minus( b ); }
          | #(MULT a=expr b=expr)       { r = a.multiply( b ); }
          | #(DIV a=expr b=expr)        { r = a.divide( b ); }
          | #(MOD a=expr b=expr)        { r = a.modulus( b ); }
          | #(UPLUS a=expr)             { r = a; }
          | #(UMINUS a=expr)            { r = a.unaryMinus(); }
          | #(PLUSEQ a=expr b=expr)     { r = ipt.assign( a, a.plus( b ));
}
          | #(MINUSEQ a=expr b=expr)    { r = ipt.assign( a, a.minus( b ));
}
          | #(MULTEQ a=expr b=expr)     { r = ipt.assign( a, a.multiply( b
)); }
```

```
        | #(DIVEQ a=expr b=expr)     { r = ipt.assign( a, a.divide( b
)); }
        | #(MODEQ a=expr b=expr)     { r = ipt.assign( a, a.modulus( b
)); }
        | #(ASGN a=expr b=expr)      { r = ipt.assign( a, b ); }

        | #(CALL fid:ID x=mexpr)
                  { r = ipt.funcInvoke( fid.getText(), x ); }

        | num:INT                    { r = ipt.getNumber( num.getText()
); }
        | str:STRING                 { r = new AweString( str.getText()
); }
        | "true"                     { r = new AweBoolean( true ); }
        | "false"                    { r = new AweBoolean( false ); }
        | #(id:ID                    { r = ipt.getVariable( id.getText()
); }
          )
        | #(DECLARE vartype:type (varid:ID       { r =
ipt.declareVariable( vartype.getText(), varid.getText() ); }) +
          )
        | #("if" a=expr thenp:. (elsep:.)?)
            {
                if ( !( a instanceof AweBoolean ) )
                    return a.throwError( "if: expression should be bool"
);
                  System.out.println("checkIfCondition " +
((AweBoolean)a).data);
                if ( ((AweBoolean)a).data )
                    r = expr( #thenp );
                else if ( null != elsep )
                    r = expr( #elsep );
            }
        | #(STATEMENT (stmt:. { r = expr(#stmt); } )*)
        ;

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

```
/*
 * AweWebfile.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

import java.io.*;
import java.net.*;
import java.util.regex.*;

public class AweWebfile extends AweDataType
{
      String Title;
      String URL;
      String MimeType;
      int Status;
      byte[] DataBytes;
      String DataString;

      public String toString()
      {
         return "*{\"" + URL + "\", \"" + Title + "\"}";
      }

      public AweWebfile()
    {
            this.Title        =     null;
      this.URL          =     null;
      this.MimeType     =     null;
      this.Status       =     -1;
      this.DataBytes          =     null;
      this.DataString         =     null;
    }
    public AweWebfile(String URL)
    {
      this.Title        =     null;
      this.URL          =     URL;
      this.MimeType     =     null;
      this.Status       =     -1;
      this.DataBytes          =     null;
      this.DataString         =     null;
      initialized       =     true;

      this.LoadContent();
    }

    public AweWebfile(String URL, String Title)
    {
      this.Title        =     Title;
      this.URL          =     URL;
      this.MimeType     =     null;
      this.Status       =     -1;
      this.DataBytes          =     null;
      this.DataString         =     null;
      initialized       =     true;

      this.LoadContent();
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweWebfile)
      {
```

```
            this.Title        =     ((AweWebfile)a).Title;
            this.URL          =     ((AweWebfile)a).URL;
            this.MimeType     =     ((AweWebfile)a).MimeType;
            this.Status       =     ((AweWebfile)a).Status;
            this.DataBytes    =     ((AweWebfile)a).DataBytes;
            this.DataString   =     ((AweWebfile)a).DataString;
            initialized       =     true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
        return "Webfile";
    }

    public AweDataType Title()
    {
      return new AweString(Title);
    }

    public AweDataType URL()
    {
      return new AweString(URL);
    }

    public void setTitle(AweDataType title)
    {
      if (title instanceof AweString)
      {
            this.Title = ((AweString)title).data;
      }
    }

    public AweDataType copy()
    {
        return new AweHyperlink( this.URL, this.Title );
    }

    private void LoadContent()
    {
      try
            {
            if (!this.URL.startsWith("http://"))
            {
                  this.URL = "http://" + this.URL;
            }

                  URL server = new URL(this.URL);
                  String content = "";

                  HttpURLConnection connection =
(HttpURLConnection)server.openConnection();
                  connection.connect();

                  if (connection.getResponseCode() == 200)
                  {
                        InputStream in = connection.getInputStream();
                        BufferedReader reader = new BufferedReader(new
InputStreamReader(in,"UTF-8"));
```

```
                        String line;

                        while ((line = reader.readLine()) != null)
                        {
                              content += line;
                        }

                        reader.close();
                  }

                  this.MimeType           =
  connection.getContentType();
                  this.Status             =
  connection.getResponseCode();
                  this.DataBytes          =     content.getBytes();
                  this.DataString         =     content;

            }
            catch (Exception e)
            {
                  throwError("Network Error");
            }
}

public AweDataType status()
{
  return new AweInteger(this.Status);
}

public AweDataType isSuccess()
{
  if(this.Status == 200)
        return new AweBoolean(true);

  return new AweBoolean(false);
}

public AweDataType mimeType()
{
  return new AweString(this.MimeType);
}

public AweDataType isHTML()
{
  if(this.MimeType.equals("text/html"))
        return new AweBoolean(true);

  return new AweBoolean(false);
}

public AweDataType html()
{
  return new AweString(this.DataString);
}

public AweDataType isImage()
{
  if(this.MimeType.startsWith("image"))
        return new AweBoolean(true);

  return new AweBoolean(false);
}

public AweDataType getHyperLinks()
```

```
{
  AweCollection ac = new AweCollection();
  String baseURL = this.URL.replaceAll("/[\\w]*\\.*[\\w]*$","");

  Pattern pattern;
        Matcher matcher;

        String regex = "href\\s*=\\s*\"*[^\">]*";

        System.out.println(regex);

        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(this.DataString);

        while(matcher.find())
        {
              String temp = matcher.group();
              String done = temp.replaceAll("href\\s*=\\s*\"*","");

              if (done.startsWith("/") || done.indexOf("/") == -1)
              {
                    done = baseURL + done;
              }

              ac.add(new AweHyperlink(done));
        }

  return ac;
}

public AweDataType getImageLinks()
{
  AweCollection ac = new AweCollection();
  String baseURL = this.URL.replaceAll("/[\\w]*\\.*[\\w]*$","");

  Pattern pattern;
        Matcher matcher;

        String regex = "src\\s*=\\s*\"*[^\">]*";

        System.out.println(regex);

        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(this.DataString);

        while(matcher.find())
        {
              String temp = matcher.group();
              String done = temp.replaceAll("src\\s*=\\s*\"*","");

              if (done.startsWith("/") || done.indexOf("/") == -1)
              {
                    done = baseURL + done;
              }

              if (done.endsWith("jpg") || done.endsWith("gif")
              || done.endsWith("jpeg") || done.endsWith("bmp")
              || done.endsWith("tiff") || done.endsWith("png")
              || done.endsWith("tga"))
              {
                    ac.add(new AweHyperlink(done));
              }
        }
```

```
      return ac;
    }

    public void writeToFile(AweDataType filePath)
      {
            String fileName;

            fileName = this.URL.replace(':','_');
            fileName = fileName.replace('/','_');
            fileName = fileName.replace('.','_');

            try
            {
                  FileOutputStream fos = new FileOutputStream(filePath +
"\\" + fileName);
                  fos.write(this.DataBytes);
            }
            catch (Exception e)
            {
                  throwError("File write error");
            }
    }
}
```

```
/*
 * AweAweSymbolResolver.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

public class AweSymbolResolver extends AweDataType
{
      public static String StatusString = null;

      public static AweDataType funcInvoke( String func, AweDataType[]
params ) //throws antlr.RecognitionException {
      {
            if (func.equals("url"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for url";
                        return null;
                  }

                  if (params[0] instanceof AweHyperlink)
                  {
                        StatusString = null;
                        return new
AweString(((AweHyperlink)params[0]).URL);
                  }

                  StatusString = "Incorrect type of parameters for url";
                  return null;
            }
            else if (func.equals("text"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for text";
                        return null;
                  }

                  if (params[0] instanceof AweHyperlink)
                  {
                        StatusString = null;
                        return new
AweString(((AweHyperlink)params[0]).Title);
                  }

                  StatusString = "Incorrect type of parameters for
text";
                  return null;
            }
            else if (func.equals("count"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for count";
                        return null;
                  }

                  if (params[0] instanceof AweCollection)
                  {
```

```
                        StatusString = null;
                        return ((AweCollection)params[0]).count();
                  }

                  StatusString = "Incorrect type of parameters for
count";
                  return null;
            }
            else if (func.equals("add"))
            {
                  if (params.length != 2)
                  {
                        StatusString = "Incorrect number of parameters
for add";
                        return null;
                  }

                  if (params[0] instanceof AweCollection && params[1]
instanceof AweHyperlink)
                  {
                        StatusString = null;
                        return
((AweCollection)params[0]).add((AweHyperlink)params[1]);
                  }

                  StatusString = "Incorrect type of parameters for add";
                  return null;
            }
            else if (func.equals("remove"))
            {
                  if (params.length != 2)
                  {
                        StatusString = "Incorrect number of parameters
for remove";
                        return null;
                  }

                  if (params[0] instanceof AweCollection && params[1]
instanceof AweHyperlink)
                  {
                        StatusString = null;
                        return
((AweCollection)params[0]).remove((AweHyperlink)params[1]);
                  }

                  StatusString = "Incorrect type of parameters for
remove";
                  return null;
            }
            else if (func.equals("clear"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for clear";
                        return null;
                  }

                  if (params[0] instanceof AweCollection)
                  {
                        StatusString = null;
                        ((AweCollection)params[0]).clear();
                  }
```

```
                  StatusString = "Incorrect type of parameters for
clear";
                  return null;
            }
            else if (func.equals("status"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for status";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).status();
                  }

                  StatusString = "Incorrect type of parameters for
status";
                  return null;
            }
            else if (func.equals("isSuccess"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for isSuccess";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).isSuccess();
                  }

                  StatusString = "Incorrect type of parameters for
isSuccess";
                  return null;
            }
            else if (func.equals("isHTML"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for isHTML";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).isHTML();
                  }

                  StatusString = "Incorrect type of parameters for
isHTML";
                  return null;
            }
            else if (func.equals("isImage"))
            {
                  if (params.length != 1)
```

```
                  {
                        StatusString = "Incorrect number of parameters
for isImage";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).isImage();
                  }

                  StatusString = "Incorrect type of parameters for
isImage";
                  return null;
            }
            else if (func.equals("type"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for type";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).mimeType();
                  }

                  StatusString = "Incorrect type of parameters for
type";
                  return null;
            }
            else if (func.equals("html"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for html";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).html();
                  }

                  StatusString = "Incorrect type of parameters for
html";
                  return null;
            }
            else if (func.equals("links"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for links";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
```

```
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).getHyperLinks();
                  }

                  StatusString = "Incorrect type of parameters for
links";
                  return null;
            }
            else if (func.equals("images"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for images";
                        return null;
                  }

                  if (params[0] instanceof AweWebfile)
                  {
                        StatusString = null;
                        return ((AweWebfile)params[0]).getImageLinks();
                  }

                  StatusString = "Incorrect type of parameters for
images";
                  return null;
            }
            else if (func.equals("print"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for print";
                        return null;
                  }

                  if (params[0] instanceof AweDataType)
                  {
                        StatusString = null;
                        ((AweDataType)params[0]).print();
                        return null;
                  }

                  StatusString = "Incorrect type of parameters for
print";
                  return null;
            }
            else if (func.equals("save"))
            {
                  if (params.length != 2)
                  {
                        StatusString = "Incorrect number of parameters
for save";
                        return null;
                  }

                  if (params[0] instanceof AweString && params[1]
instanceof AweString)
                  {
                        StatusString = null;

      ((AweString)params[0]).save((AweString)params[1]);
                  }
```

```
                  StatusString = "Incorrect type of parameters for
save";
                  return null;
            }
            else if (func.equals("indexOf"))
            {
                  if (params.length != 3)
                  {
                        StatusString = "Incorrect number of parameters
for indexOf";
                        return null;
                  }

                  if (params[0] instanceof AweString && params[1]
instanceof AweString && params[2] instanceof AweInteger)
                  {
                        StatusString = null;
                        return
((AweString)params[0]).indexOf((AweString)params[1],(AweInteger)params[2
]);
                  }

                  StatusString = "Incorrect type of parameters for
indexOf";
                  return null;
            }
            else if (func.equals("length"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for length";
                        return null;
                  }

                  if (params[0] instanceof AweString)
                  {
                        StatusString = null;
                        return ((AweString)params[0]).length();
                  }

                  StatusString = "Incorrect type of parameters for
length";
                  return null;
            }
            else if (func.equals("replace"))
            {
                  if (params.length != 3)
                  {
                        StatusString = "Incorrect number of parameters
for replace";
                        return null;
                  }

                  if (params[0] instanceof AweString && params[1]
instanceof AweString && params[2] instanceof AweString)
                  {
                        StatusString = null;
                        return
((AweString)params[0]).replace((AweString)params[1],(AweString)params[2]
);
                  }
```

```
                  StatusString = "Incorrect type of parameters for
replace";
                  return null;
            }
            else if (func.equals("subString"))
            {
                  if (params.length != 3)
                  {
                        StatusString = "Incorrect number of parameters
for subString";
                        return null;
                  }

                  if (params[0] instanceof AweString && params[1]
instanceof AweInteger && params[2] instanceof AweInteger)
                  {
                        StatusString = null;
                        return
((AweString)params[0]).subString((AweInteger)params[1],(AweInteger)param
s[2]);
                  }

                  StatusString = "Incorrect type of parameters for
subString";
                  return null;
            }
            else if (func.equals("getItemByIndex"))
            {
                  if (params.length != 2)
                  {
                        StatusString = "Incorrect number of parameters
for getItemByIndex";
                        return null;
                  }

                  if (params[0] instanceof AweCollection && params[1]
instanceof AweInteger)
                  {
                        StatusString = null;
                        return
((AweCollection)params[0]).getAtIndex((AweInteger)params[1]);
                  }

                  StatusString = "Incorrect type of parameters for
getItemByIndex";
                  return null;
            }
            else if (func.equals("findItemIndex"))
            {
                  if (params.length != 2)
                  {
                        StatusString = "Incorrect number of parameters
for findItemIndex";
                        return null;
                  }

                  if (params[0] instanceof AweCollection && params[1]
instanceof AweString)
                  {
                        StatusString = null;
                        return
((AweCollection)params[0]).findIndex((AweString)params[1]);
                  }
```

```
                  StatusString = "Incorrect type of parameters for
findItemIndex";
                  return null;
            }
            else if (func.equals("toString"))
            {
                  if (params.length != 1)
                  {
                        StatusString = "Incorrect number of parameters
for toString";
                        return null;
                  }

                  if (params[0] instanceof AweDataType)
                  {
                        StatusString = null;
                        return new AweString(params[0].toString());
                  }

                  StatusString = "Incorrect type of parameters for
toString";
                  return null;
            }
            else
            {
                  StatusString = "No such function";
                  return null;
            }
    }
}
```

```
/*
 * AweString.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

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

class AweString extends AweDataType
{
    String data;

    public String toString()
    {
      return data;
    }

    public AweString()
    {
        data = null;
    }

    public AweString(String init)
    {
        this.data             =     init;
        initialized     =     true;
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweString)
      {
            this.data = ((AweString)a).data;
            initialized = true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
        return "String";
    }

    public AweDataType copy()
    {
        return new AweString( data );
    }

    public AweDataType stringConcatenate( AweDataType a )
    {
        if (a instanceof AweString)
            return new AweString( this.data + ((AweString)a).data );

        return throwError ("^", a);
    }

    public AweDataType equal( AweDataType a )
    {
```

```
    if ( a instanceof AweBoolean )
        return new AweBoolean( this.data == ((AweString)a).data );

    return throwError("==", a);
}

public AweDataType notEqual( AweDataType a )
{
    if ( a instanceof AweBoolean )
        return new AweBoolean( this.data != ((AweString)a).data );

    return throwError("!=", a);
}

public AweDataType regexEqual( AweDataType a )
{
    Pattern pattern;
    Matcher matcher;

    if (a instanceof AweString)
    {
        pattern = Pattern.compile(((AweString)a).data);
        matcher = pattern.matcher(this.data);

        boolean found = false;

        while(matcher.find())
        {
              found = true;
        }

        return new AweBoolean(found);
    }

    return throwError ("~~", a);
}

public AweDataType regexNotEqual( AweDataType a )
{
    Pattern pattern;
    Matcher matcher;

    if (a instanceof AweString)
    {
        pattern = Pattern.compile(((AweString)a).data);
        matcher = pattern.matcher(this.data);

        boolean found = true;

        while(matcher.find())
        {
              found = false;
        }

        return new AweBoolean(found);
    }

    return throwError ("!~", a);
}

public void save(AweDataType fileName)
{
  try
  {
```

```
            if (fileName instanceof AweString)
            {
                  FileOutputStream out;
                  PrintStream p;

                  out = new FileOutputStream(AweConstants.HomeDirectory
+ ((AweString)fileName).data);
                  p = new PrintStream( out );
                  p.print(this.data);
            }
      }
      catch (Exception e)
      {

      }
    }

    public AweDataType indexOf (AweDataType string1, AweDataType start)
    {
            if ((string1 instanceof AweString) && (start instanceof
AweInteger))
        {
                  int index =
(this.data.indexOf(((AweString)string1).data,
((AweInteger)start).data));
                  return new AweInteger(index);
        }

            return new AweInteger(-1);
    }

    public AweDataType length()
    {

      if (this.data != null)
      {
            return new AweInteger(this.data.length());
      }

      return new AweInteger(-1);
    }

    public AweDataType replace(AweDataType string1, AweDataType string2)
    {
            if ((string1 instanceof AweString) && (string2 instanceof
AweString))
        {
                  return new
AweString(this.data.replaceFirst(((AweString)string1).data,((AweString)s
tring2).data));
        }

            return new AweString(this.data);
    }

    public AweDataType subString(AweDataType start, AweDataType length)
    {
            if ((start instanceof AweInteger) && (length instanceof
AweInteger))
        {
                  return new
AweString(this.data.substring(((AweInteger)start).data,((AweInteger)leng
th).data));
        }
```

```
            return new AweString(this.data);
        }

    }
```

```
/*
 * AweInterpreter.java
 *
 * @author Khai Quang Vuong - kqv1@columbia.edu
 *
 */


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

public class AweInterpreter {
      Hashtable varList = new Hashtable();

      public AweInterpreter() {
      }

      //Fake function, used for testing only
    public AweDataType op(String action, String object)
    {
      System.out.println(action + " " + object);
      return new AweDataType("Test");
    }

    public static AweDataType getNumber( String s ) {
      System.out.println(";          getNumber " + s);
      AweInteger i = new AweInteger( Integer.parseInt( s ) );
        return i;
    }

    public AweDataType getVariable( String var ) {
      AweDataType x = (AweDataType)varList.get(var);
      if (x == null)
            throw new AweException("Unknown variable");
      System.out.println(";          getVariable " + var + " value " +
x.toString());

      return x;
    }

    public AweDataType declareVariable( String type, String var ) {
      System.out.println(";          declareVariable " + var + " of " +
type);
      AweDataType x;
      if (type.equals("int"))
            x = new AweInteger();
      else if (type.equals("bool"))
            x = new AweBoolean();
      else if (type.equals("string"))
            x = new AweString();
      else if (type.equals("webfile"))
            x = new AweWebfile();
      else if (type.equals("hyperlink"))
            x = new AweHyperlink();
      else if (type.equals("collection"))
            x = new AweCollection();
      else
            throw new AweException("Unsupport type");
      varList.put(var, x);
```

```
        return x;
    }

    public AweDataType assign( AweDataType a, AweDataType b ) {
      System.out.println(";          assign " + a.typeName() + " to " +
b.toString());

      a.assign(b);

      return a;
    }

    public boolean canProceed() {
        return true;
    }

    public AweDataType funcInvoke(
            String func,
            AweDataType[] params ) throws antlr.RecognitionException {
      System.out.println(";          funcInvoke " + func);
      AweDataType x = AweSymbolResolver.funcInvoke(func, params);
      if (AweSymbolResolver.StatusString != null)
            throw new AweException(AweSymbolResolver.StatusString);
      return x;
    }

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

```
/*
 * AweInteger.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

class AweInteger extends AweDataType
{
    int data;

    public String toString()
      {
         return Integer.toString(data);
      }

    public AweInteger()
    {

    }

    public AweInteger(int init)
    {
        data = init;
        initialized = true;
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweInteger)
      {
            this.data = ((AweInteger)a).data;
            initialized = true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
        return "Integer";
    }

    public AweDataType copy()
    {
        return new AweInteger( this.data );
    }

    public static int getIntegerValue(AweDataType a)
    {
//          if ( a instanceof AweDouble )
//                return (int) ((AweDouble)a).data;
        if ( a instanceof AweInteger )
            return ((AweInteger)a).data;
        if ( a instanceof AweBoolean )
        {
            if (((AweBoolean)a).data == true)
                  return 1;
            else
                  return 0;
        }
```

```
        a.throwError("Cannot cast to Integer");
        return 0;
    }

    public AweDataType toAweString()
    {
      return new AweString(Integer.toString(data));
    }

    public AweDataType complementNumber()
    {
        return new AweInteger(-this.data);
    }

    public AweDataType unaryMinus()
    {
        return new AweInteger(--this.data);
    }

    public AweDataType unaryPlus()
    {
        return new AweInteger(++this.data);
    }

    public AweDataType plus(AweDataType a)
    {
//        if (a instanceof AweInteger)
          return new AweInteger(this.data +
AweInteger.getIntegerValue(a));
//        return new AweDouble(this.data + AweDouble.getDoubleValue(a));
    }

    public AweDataType minus(AweDataType a)
    {
//        if (a instanceof AweInteger)
          return new AweInteger( this.data -
AweInteger.getIntegerValue(a));
//        return new AweDouble( this.data -
AweDouble.getDoubleValue(a));
    }

    public AweDataType multiply(AweDataType a)
    {
//        if (a instanceof AweInteger)
          return new AweInteger( this.data *
AweInteger.getIntegerValue(a));
//        return new AweDouble( this.data *
AweDouble.getDoubleValue(a));
    }

    public AweDataType divide(AweDataType a)
    {
//        if (a instanceof AweInteger)
          return new AweInteger( this.data /
AweInteger.getIntegerValue(a));
//        return new AweDouble( this.data /
AweDouble.getDoubleValue(a));
    }

    public AweDataType modulus(AweDataType a)
    {
//        if (a instanceof AweInteger)
          return new AweInteger( this.data %
AweInteger.getIntegerValue(a));
```

```
//        return new AweDouble( this.data %
AweDouble.getDoubleValue(a));
    }


    public AweDataType greaterThan( AweDataType a )
    {
      return new AweBoolean( this.data > AweInteger.getIntegerValue(a));
    }

    public AweDataType lessThan( AweDataType a )
    {
      return new AweBoolean( this.data < AweInteger.getIntegerValue(a));
    }

    public AweDataType greaterThanEqual( AweDataType a )
    {
      return new AweBoolean( this.data >=
AweInteger.getIntegerValue(a));
    }

    public AweDataType lessThanEqual( AweDataType a )
    {
      return new AweBoolean( this.data <= AweInteger.getIntegerValue(a)
);
    }

    public AweDataType equal( AweDataType a)
    {
      return new AweBoolean(this.data == AweInteger.getIntegerValue(a)
);
    }

    public AweDataType notEqual( AweDataType a)
    {
      return new AweBoolean(this.data != AweInteger.getIntegerValue(a));
    }
}
```

```
/*
 * AweHyperlink.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

public class AweHyperlink extends AweDataType
{
      String Title;
      String URL;

      public String toString()
      {
         return "{\"" + URL + "\", \"" + Title + "\"}";
      }

      public AweHyperlink()
    {
      Title       = null;
      this.URL    = null;
    }

    public AweHyperlink(String URL)
    {
      Title       = null;
      this.URL    = URL;
      initialized = true;
    }

    public AweHyperlink(String URL, String Title)
    {
      this.Title  = Title;
      this.URL    = URL;
      initialized = true;
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweHyperlink)
      {
            this.Title        =     ((AweHyperlink)a).Title;
            this.URL          =     ((AweHyperlink)a).URL;
            initialized       =     true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
        return "Hyperlink";
    }

    public AweDataType Title()
    {
      return new AweString(Title);
    }

    public AweDataType URL()
    {
      return new AweString(URL);
```

```
    }

    public void setTitle(AweDataType title)
    {
      if (title instanceof AweString)
      {
            this.Title = ((AweString)title).data;
      }
    }

    public void setURL(AweDataType url)
    {
      if (url instanceof AweString)
      {
            this.URL = ((AweString)url).data;
      }
    }

    public void setHyperlink(AweDataType url, AweDataType title)
    {
      if ((url instanceof AweString) && (title instanceof AweString))
      {
            this.URL = ((AweString)url).data;
            this.Title = ((AweString)title).data;
      }
    }

    public AweDataType copy()
    {
       return new AweHyperlink( this.URL, this.Title );
    }
}
```

```java
/*
 * AweException.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

public class AweException extends RuntimeException
{
	static final long serialVersionUID = 564987654654L;

	AweException( String exc )
	{
		System.err.println("An exception occured: " + exc);
	}
}
```

```
/*
 * AweDataType.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

public class AweDataType
{
    private String typeName;
    protected boolean initialized;

    public String toString()
    {
      return typeName;
    }

    public AweDataType()
    {
      typeName = null;
      initialized = false;
    }

    public boolean initialized()
    {
      return initialized;
    }

    public void print()
    {
            System.out.println(this.toString());
    }

    public AweDataType(String name)
    {
        this.typeName = name;
    }

    public String typeName()
    {
        return "UnknownType";
    }

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

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

    public AweDataType throwError(String symbol, AweDataType a)
    {
      String exc = "Error with operation: " + symbol
                          +  " with type " + ( a.typeName != null ?
a.typeName : "UnknownType" );
        throw new AweException(exc);
    }

    public AweDataType throwError(String symbol)
    {
```

```
  String exc = "Error with operation: " + symbol;
    throw new AweException(exc);
}

public void assign( AweDataType a )
{
    throwError("=", a);
}

public AweDataType notEqual( AweDataType a )
{
    return throwError("!=", a);
}

public AweDataType equal( AweDataType a )
{
    return throwError("==", a);
}

public AweDataType greaterThanEqual( AweDataType a )
{
    return throwError(">=", a);
}

public AweDataType lessThanEqual( AweDataType a )
{
    return throwError(">=", a);
}

public AweDataType plus( AweDataType a )
{
    return throwError("+", a);
}

public AweDataType minus( AweDataType a )
{
    return throwError("-", a);
}

public AweDataType multiply( AweDataType a )
{
    return throwError("*", a);
}

public AweDataType divide( AweDataType a )
{
    return throwError("/", a);
}

public AweDataType modulus( AweDataType a )
{
    return throwError("%", a);
}

public AweDataType booleanEqual( AweDataType a )
{
    return throwError("&&", a);
}

public AweDataType booleanOr( AweDataType a )
{
    return throwError("||", a);
}
```

```
    public AweDataType unaryPlus()
    {
        return throwError("++");
    }

    public AweDataType unaryMinus()
    {
        return throwError("--");
    }

    public AweDataType greaterThan( AweDataType a )
    {
        return throwError(">", a);
    }

    public AweDataType lessThan( AweDataType a )
    {
        return throwError("<", a);
    }

    public AweDataType stringConcatenate( AweDataType a )
    {
        return throwError("^", a);
    }

    public AweDataType makePointer( AweDataType a )
    {
        return throwError("*", a);
    }

    public AweDataType makeReversePointer( AweDataType a )
    {
        return throwError("&", a);
    }

    public AweDataType makePipe( AweDataType a )
    {
        return throwError("->", a);
    }

    public AweDataType regexEqual( AweDataType a )
    {
        return throwError("~~", a);
    }

    public AweDataType regexNotEqual( AweDataType a )
    {
        return throwError("!~", a);
    }

    public AweDataType complementBoolean()
    {
        return throwError("!");
    }

    public AweDataType complementNumber()
    {
        return throwError("-");
    }
}
```

```
/*
 * AweConstants.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

public class AweConstants
{
      public static String HomeDirectory = "C:\\AweRuntime\\";
}
```

```
/*
 * AweCollection.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */


import java.util.*;

public class AweCollection extends AweDataType
{
      ArrayList hyperlinkCollection;

      public String toString()
      {
         return hyperlinkCollection.toString();
      }

    public AweCollection()
    {
      hyperlinkCollection     =     new ArrayList();
      initialized                   =     true;
    }

    public AweCollection(ArrayList Collection)
    {
      this.hyperlinkCollection      =     Collection;
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweCollection)
      {
            this.hyperlinkCollection =
((AweCollection)a).hyperlinkCollection;
            initialized = true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
       return "Collection";
    }

    public AweDataType copy()
    {
       return new AweCollection(this.hyperlinkCollection);
    }

    public AweDataType add(AweDataType hyperlink)
    {
      if (hyperlink instanceof AweHyperlink)
      {
            hyperlinkCollection.add(hyperlink);
            return new AweInteger(hyperlinkCollection.size() - 1);
      }

      return new AweInteger(-1);
    }
```

```
    public AweDataType remove(AweDataType hyperlink)
    {
      if (hyperlink instanceof AweHyperlink)
      {
            if (hyperlinkCollection.contains(hyperlink))
            {
                  hyperlinkCollection.remove(hyperlink);
                  return new AweBoolean(true);
            }
      }

      return new AweBoolean(false);
    }

    public AweDataType count()
    {
      return new AweInteger(hyperlinkCollection.size());
    }

    public void clear()
    {
      hyperlinkCollection.clear();
    }

    public AweDataType getAtIndex(AweDataType index)
    {
      if(index instanceof AweInteger && ((AweInteger)index).data <
hyperlinkCollection.size() )
      {
            return
(AweDataType)hyperlinkCollection.get(((AweInteger)index).data);
      }

      return throwError("Index out of Bounds!");
    }

    public AweDataType findIndex(AweDataType linkText)
    {
      if(linkText instanceof AweString)
      {
            for (int i = 0; i < hyperlinkCollection.size(); i++)
            {
                  if
(((AweHyperlink)hyperlinkCollection.get(i)).URL.equals(((AweString)linkT
ext).data))
                  {
                        return new AweInteger(i);
                  }
            }
      }

      return new AweInteger(-1);
    }
}
```

```
/*
 * AweBoolean.java
 *
 * @author George Sirois - gts2006@columbia.edu
 *
 */

class AweBoolean extends AweDataType
{
    boolean data;

    public String toString()
    {
      return Boolean.toString(data);
    }

    AweBoolean(  )
    {

    }

    AweBoolean( boolean init )
    {
        this.data = init;
        initialized = true;
    }

    public void assign(AweDataType a)
    {
      if(a instanceof AweBoolean)
      {
            this.data = ((AweBoolean)a).data;
            initialized = true;
      }
      else
      {
            throwError("=", a);
      }
    }

    public String typeName()
    {
        return "Boolean";
    }

    public AweDataType copy()
    {
        return new AweBoolean(this.data);
    }

    public AweDataType toAweString()
    {
      return new AweString(Boolean.toString(data));
    }

    public AweDataType booleanEqual( AweDataType a )
    {
        if ( a instanceof AweBoolean )
            return new AweBoolean( this.data && ((AweBoolean)a).data );

        return throwError("&&", a);
    }

    public AweDataType booleanOr( AweDataType a )
```

```
    {
        if ( a instanceof AweBoolean )
            return new AweBoolean( this.data || ((AweBoolean)a).data );

        return throwError("||", a);
    }

    public AweDataType complementBoolean()
    {
        return new AweBoolean(!this.data);
    }

    public AweDataType equal( AweDataType a )
    {
        if ( a instanceof AweBoolean )
            return new AweBoolean( this.data == ((AweBoolean)a).data );

        return throwError("==", a);
    }

    public AweDataType notEqual( AweDataType a )
    {
        if ( a instanceof AweBoolean )
            return new AweBoolean( this.data != ((AweBoolean)a).data );

        return throwError("!=", a);
    }
}
```