COMS W4115

Programming Languages and Translators

Prof. Stephen Edwards

Final Report

MASC

Fall 2007

Christos Angelopoulos
Kristina Chodorow
Michael Masullo
Vaibhav Saharan
{ca2269,kc2424,msm2148,vs2220}@columbia.edu

1. Whitepaper

MASC is a language for creating fast information extraction tools. The MASC shell is embedded in the web-browser and allows users to write code snippets in MASC while browsing the web. It offers the user a number of interesting features to process the data on the web-page, like crawling the web-page, extracting various types of information, saving different types of media etc. Furthermore, it also allows the user to author general purpose stand-alone programs. This is as important component of the language as it gives the user, the ability to process the data available on web page in a faster and more efficient manner. It is more stable, concise, and occupies less memory than existing GUI tools. MASC helps to work with the information available on the web in a smarter way.

Consider some examples to this end. Search results often return web-pages riddled with advertisements and irrelevant data. MASC can help derive the relevant information and meaningful links from such results. Alternatively, it can help crawl a web page to find links that go to relevant results and even save such results for future reference. It can help save the various media elements like images, sound snippets, videos etc., embedded in a web page. Such uses of MASC abound. This document, as the name suggests, serves as a way to introduce the user to the features and constructs of this language and also offers a few examples in an attempt to exhibit the use of some of these features.

2. Tutorial

Welcome to MASC, the web-browser based shell language that helps you to organize the data that appears in your browser, as well as run some stand-alone programs. This tutorial is designed to help you familiarize yourself with what MASC does and how you can use the language to its full capabilities.

2.1 The MASC Shell

The text box at the bottom of your browser with the >>> in it is the MASC shell. It's where all the magic happens. All your programs will be written here, and, unless you say otherwise, the results will be put here. You can also use it to control your browser to some extent. Try typing in the following and then pressing enter:

```
echo "hi"
```

MASC will say hi right back, if only because you told it to, using the shell command "echo." It's not exactly Hello World, but it demonstrates the point that it needs to make: the MASC shell can do most shell commands.

2.2 Getting Started

Let's say you're using a Linux distribution and you want to look at all the bookmarks you have. The first thing you want to do is assign your bookmarks file to a variable. Variables in MASC are typeless, so you can easily have a number or a string or a filename (which is essentially a string, anyway) without declaring what exactly it is you're assigning to it. So if your bookmarks file is located at /home/user/.bookmarks you can assign that filename to a variable by typing in

```
bmarks_file="/home/user/.bookmarks"
```

Semicolons are optional in MASC as long as you press enter after each command. Since the MASC shell is an interpreter, your command is interpreted right away and bmarks_file now has your bookmarks file in it. Of course, that's only half the battle. Now you need to tell MASC to display your bookmarks, which is done using a shell command. Type in

```
cat bmarks_file | sort
```

The cat command lets you display the file contained in bmarks_file, while the | after it pipes the file to the sort command, which sorts the bookmarks and then displays it in the MASC shell. Essentially, this command works just like it would if you were using a shell. (NOTE: not all MASC commands are the same as shell commands. If you wanted to redirect the sorted bookmarks to a file instead of displaying them on the screen in the MASC shell you would type

```
cat bmarks_file | sort -> mybmarks.txt
```

Notice that the redirection operator is "->" instead of >. > in MASC is reserved for the "greater than" operator. For a further listing of MASC operations, see later on in this document.)

2.3 Loops and conditionals

MASC supports while loops and if statements, but if statements have a bit of a different syntax than what you're probably used to, so let's see an example to clear things up. Let's do an alternate take on looking at bookmarks. Let's say you're doing research and want to remember some websites. First, make an array to hold your links in:

```
array goodLinks[5]
```

That gives you 5 spots to put links in, the array keyword before goodLinks lets MASC know you're declaring an array. Now initialize it:

```
j=0
while (j<5) {goodLinks[j]=""; j=j+1}
```

The first line makes a new variable j and sets it equal to zero. The second line is a while loop that sets all the elements in goodLinks to empty strings. Note the semicolon between the two commands is required to separate them since they're on the same line (to save space). Now, reset j and add websites as we need them:

```
j=0
newSite="www.televisionwithoutpity.com"
while(j<5){
if(goodLinks[j]==""){
goodLinks[j]=newSite; j=0; break;
else {
if(goodLinks[j]==newSite) {
echo "Already Entered"; j=0; break; else {j=j+1}
}}}
```

The first two lines should be pretty straight forward by now; they're simple assignments. The third line brings another while loop that makes its way through the array. The if statement then checks if the position in the array is empty, and if so, puts the site in the array and breaks out of the while loop. If it's not, then it checks if the site is already in that position, says so, and then breaks out. Otherwise it just increments j and moves on. Again, the important thing to note here is the syntax of the if/else. Every else is *inside* the if it is associated with. If you can understand that, and know your shell commands, you're well on your way to becoming an expert in MASC.

2. General Lexical Conventions

Any given MASC token, falls into one of these broad categories: identifiers, keywords, constants, strings, operators and general separators. With regards to these general separators, they include the likes of spaces, tabs, newline characters and comments, which are generally ignored by the compiler, with the exception that they serve to separate tokens. As a general rule, one of these characters are required to separate any two adjacent identifiers and constants.

2.1 Comments

A comment in MASC begins with a (: and ends with a corresponding :)

2.2 Identifiers

Identifiers are defined more broadly in MASC than in C-like languages. An identifier can be a variable, but it can also be a UNIX-style filename. Thus, an identifier is at least one character, possibly a "/", number, or letter. If it is more than one character, it can start with "." or "~". "var", "1/23", "/opt/bin/", "./12.3/12", and "../../" are examples of identifiers.

2.3 Keywords

MASC reserves the following identifiers as keywords to represent specific pre-defined functionalities:

for if else return continue break array TRUE FALSE

2.4 Constants

MASC has one type of constant: strings. Strings constants consist of a sequence of characters surrounded by a pair of double quotes, for example, "jack". Numbers are identifiers, not constants, as "5" may refer to a file called "5", not a number.

3. Data Types

There are no data types in MASC, any identifier can be used to represent any kind of data and can be changed to represent any other type of data. Unless otherwise noted, all operations follow standard mathematical rules when a variable is holding a number.

4. Expressions

The following is a list of expression operators, in order of precedence, from greatest to least. Within each subsection, operators have the same precedence and associativity is listed with each operator.

4.1 Primary Expressions

Primary expressions normally group from left to right.

4.1.1 *identifier*

An identifier is a primary expression, as long as it has been correctly declared.

4.1.2 *constant*

A single character or multiple characters contained in double quotes like “x” or “supercalifragilisticexpialidocious” is considered a primary expression.

4.2 Unary Operators

These operators are right associative.

4.2.1 !*expression*

The logical NOT operator reverses a number's bits or flips a boolean value.

4.2.2 -*expression*

The unary minus operator provides the additive inverse of the arithmetic value immediately to its right.

4.3 Multiplicative operators

These operators group from left to right and only work on numerical values.

4.3.1 *expression * expression*

The binary * operator multiplies the expression on the left of the * by the expression on the right. A string in either expression results in an error.

4.3.2 *expression* \\ *expression*

The binary \\ operator divides the expression on the left by the expression on the right. Once again, a string in either expression results in a syntax error, while both expressions must be integers for the result to be an integer. Otherwise, the result is a float.

4.4 Additive Operators

These operators also group from left to right.

4.4.1 *expression + expression*

The result is the sum of the two expressions if neither expression is a string. The sum will be returned as a float if both expressions are floats and as an integer if one or both of the expressions is an int. If both expressions are strings, the binary + operator becomes a concatenation operator and returns a string that is the concatenation of both expressions.

4.4.2 *expression – expression*

The binary – operator returns the difference of the two expressions. The result will be a a float unless both expressions are integers, in which case an integer is returned. A string in either expression results in a syntax error.

4.5 Relational Operators

Two expressions can be compared using <, >, <=, >=, ==, or !=. This returns a boolean value. This comparison can be useful for alphabetizing strings as the result will be based on ASCII values of the first unique character of each string.

4.7 *expression && expression*

This is the logical AND operator. It returns TRUE if both expressions are non-zero (or true) and FALSE otherwise. The && operator is lazy and operates left to right. Note that this is not a bitwise comparison.

4.8 *expression || expression*

This is the logical OR operator, which returns FALSE if both expressions are false or zero and true otherwise. The second expression is not evaluated if the first expression evaluates to TRUE something other than 0. Like & above, the || operator operates left to right, and is NOT a bitwise comparison.

4.9 *identifier = expression*

The assignment operator places the result of the expression on the right hand side of the = operator into the identifier on the left hand side of it. An identifier can also be used in place of an expression to copy the value of the right hand identifier to the left hand identifier.

4.10 Redirection Operations

4. Redirection operators:

Output Redirection

command -> file

This is equivalent to the UNIX form command > file.

command >> file

This operator appends the result of the command to the end of the file.

Input Redirection

command <- *file*

This is equivalent to the UNIX form command < file.

4.10.2. *command | command*

This operator, the pipe operator, pipes anything in stdout from the first command to the stdin of the second command.

5. Declarations

5.1 Function Declarations

All functions must be declared and defined at the same time. Therefore, the form that functions take is:

function *identifier* (*identifier* [, *identifier*]*) *compound-command*

It should be noted that MASC also supports void functions, which can be done by using the return keyword without an expression. Also, any function can contain 0 or more arguments, but all arguments must be identifiers. Multiple arguments must be separated by commas.

5.2 Array Declarations

Arrays in MASC are declared using the array keyword. That is:

array *identifier*[*expression*]

Once an array is declared, its elements can be accessed using braces such as lifeTheUniverseAndEverything[42].

6. Commands

Since MASC is a shell scripting language, the basic block is a command, rather than a statement.

6.1 Expression command

This is the most common type of command and has the form
expression;
where the semicolon serves as an optional command terminator. A newline will also terminate a command. The semicolon is only required when there are multiple commands on one line:
expression; expression; expression (: final expression does not need a semicolon:)

6.2 Conditional command

There are two forms of the conditional command:
6.3.1 Elseless

if (*expression*) { (*command*)* }

First, the expression in parentheses is evaluated. If the result is non-zero or true, the compound command is executed. Otherwise, control proceeds to the next command.
6.3.2 Else

if (*expression*) { (*command*)* else (*command*)*}

First, the expression in parentheses is evaluated. If the result is non-zero or true, the command list is executed and everything from the else onward is ignored. Otherwise, control is given to the compound command after the else.

Note that this syntax is ssomewhat different from a standard if/else syntax that you would find in other languages in that the if's brace does not close until the end of the conditional.

6.4 While command

While commands have the form:
while (*expression*) (*command)**
The command list is executed as long as the expression in parentheses is non-zero. The expression is evaluated each time before the command list is executed.

6.5 Return command

Return commands can take two forms:
return;
OR
return (*expression*);
In the first form, nothing is returned to the function that called it. An error will result if the calling function is expecting something. In the second form, the expression in parentheses is returned to the calling function.

6.6 Null command

A null commands is nothing but a semicolon like so:
;
It can be used for clarity (althogh not required) for an if or while command with no body.

7. Scope Rules

A variable exists after it is declared within the current block. If there are nested blocks, it exists in any sub-blocks.

10. Project Plan

10.1 The Making of MASC

MASC grew out of a series of very boring and unproductive discussions in early September that went something like, “What do you want to do?” “I don’t know. What do you want to do?” Eventually, we decided on doing a combination of Prolog and LISP because it was the only idea any of us had. We were all set to start putting together the proposal when an idea that would shape our group was voiced, “Hey! Why don’t we do a web-browser based language that you can use to organize content?” The idea was met with a general chorus of “That’s awesome” and “We’ll make billions,” and thus was MASC (after the initials of our last names) born.

Discussion continued with what exactly we wanted MASC to do. We came up with several idea and examples during that brainstorming session. First, we decided that if MASC was going to be effective, it need to have an interface built in to the browser. We also decided that taking on parsing HTML in addition to MASC was too much, so we found an HTML parser online and decided to hand-off most of the actual getting data from the browser to that. We came up with several other ideas that would be good for us to have in MASC, like saving all the pictures on a page to a folder or constructing a link table. We came up with a basic syntax for our language, including what symbols we wanted to use for operators, and then went off to make it work.

Of course, we still a very long way from being able to even think about how to implement all the ideas we need to do because first, we had to construct the grammar. We all knew from our brainstorming session what we wanted included in there, and soon we had a rough sketch of the grammar. We continued to tweak it, both to make it work, and as new ideas that seemed helpful popped into our heads. One of the biggest results of this was that our language ended up transforming into a shell, complete with piping and redirection. While this was going on, work also started on the tree walker and back end stuff. As the grammar made its way to its final form, work intensified on building what would become known as the MASC shell. We were initially going to have it be a Firefox plug-in, but as time passed and the deadline approached, we realized that making Firefox plug-ins run Java isn’t as easy as it sounds. Plan B was to make our own browser and put our shell in it. Believe it or not, thanks to the availability of a Java HTML renderer, this ended up being the easier choice.

Once those things were mostly done, we started testing. Most of the initial testing was done by hand. This was done mostly to debug and test a few features that we needed to make sure worked. Meanwhile, we worked on creating some test cases and writing a (bash) shell script to test them out. In short, the plan for testing was to do some stuff manually and back it up with automation. For the long version, see the test plan later in this document.

10.2 Style Guide

Antlr

If a rule consists of more than one option, place each possible option on its own line, indented with a tab and starting the line with "|".

If an Antlr rule is longer than one line, it should be broken into multiple lines, each indented a tab and a space from the left margin.

If the rule is one line long, place the semicolon at the end of the line. Otherwise, place it on its own line, indented one tab over (aligned with the "|"s above it).

Comment any rules that are not self evident, and try to break rules into logical sections.

Java

Braces opening blocks ("{") are at the end of the line describing the block:

if(x<3) {

not

if(x<3)
{

All files should be named Masc*.java. All Antlr-generated files should be named MascAntlr*.java.

Comments should be plentiful and descriptive, but they probably won't be.

Variable names should be descriptive, and to the extent that they can be without losing clarity, short. Unless a variable is temporary or a counter, it should be longer than one character. Except when absolutely necessary for clarity, names should be kept to under seven characters.

Clarity should be preferable to conciseness. No "(condition) ? thenblock : elseblock"-type forms.

Use tabs, not spaces.

Nothing in packages! It works in Eclipse, but it's a pain with Emacs/command line.

10.3 Timeline (when things were supposed to get done)

Mid September	Come up with idea
9/25	Proposal
Early October	Get feedback, discuss specifications, divide up LRM
10/18	LRM Draft, early grammar file complete
Early November	Start tree walker and back-end, investigate shell
Early December	Grammar complete and frozen
Mid December	LRM Finalized, tree walker and back-end complete

By deadline	Get MASC shell working, test and tweak, write report

10.4 Roles and Responsibilities

Christos	Static-Semantic Analysis/Back end (primary)
Kristina	Grammar (Lexer/Parser), Shell (primary), test cases
Vaibhav	Tree walker, Back end (secondary), RFT test scripting
Michael	Documentation, Shell (secondary), test cases

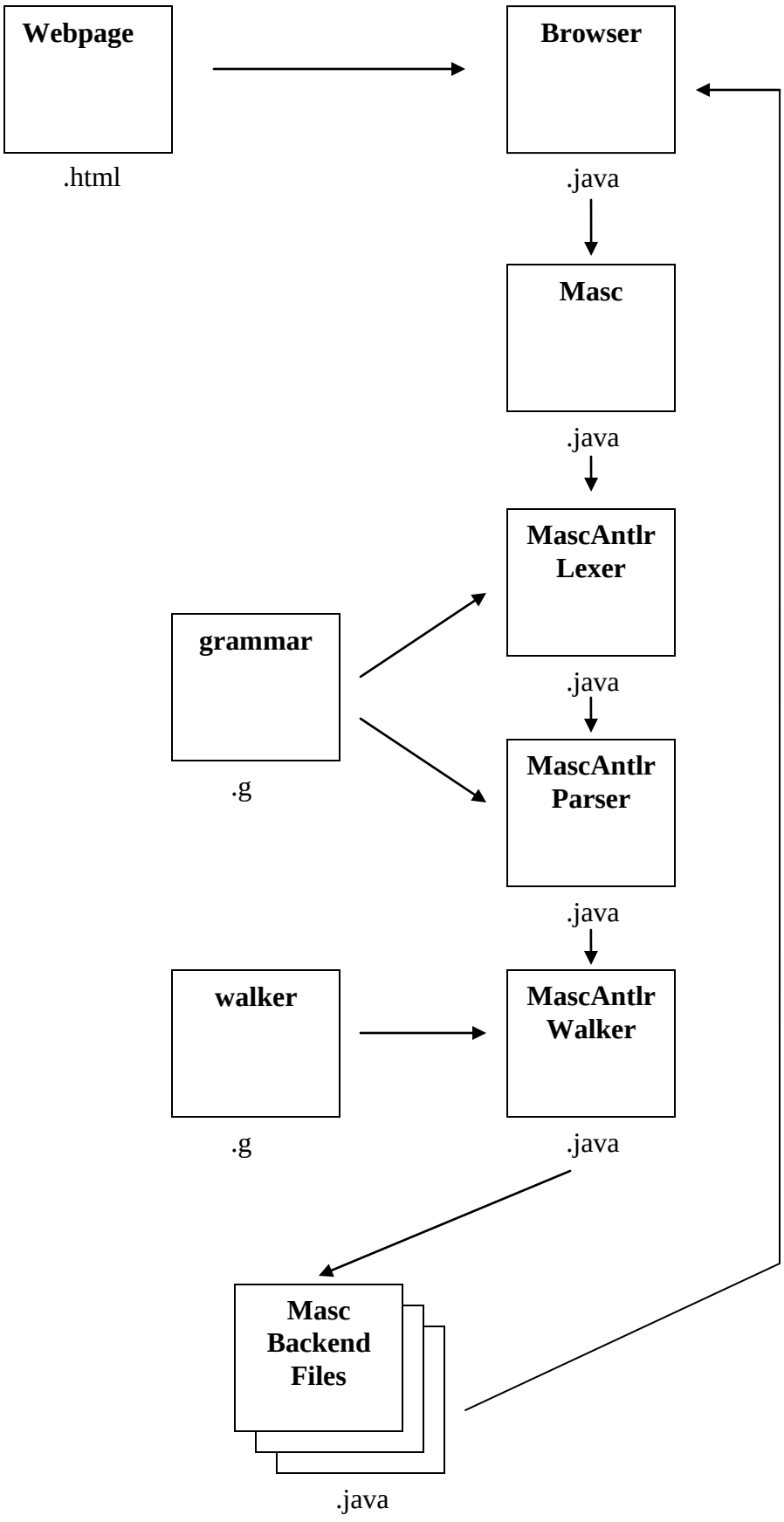
10.5 Software Development Environment

Four different people using four different computers with four different operating systems means that there were a lot of different things being used to get the job done. All of us used ANTLR 2.7.7, Java, and subversion as basic tools and programming languages. Within that, some of us also used Eclipse (and subclipse and ANTLR Eclipse plug-ins) to get things working. Emacs also was useful for some file editing. When we were thinking about using a Firefox plug-in, we stole some code from MIT's PiggyBank, which was able to do what we didn't and get JavaScript and Java to play nice in a Firefox plug-in. Eventually, we just used loboBrowser as our Java HTML Renderer to create the MASC Browser with the MASC shell inside.

10.6 Project Log

9/23	Basic Specifications Agreed on
10/5	Google Code site set up
10/17	LRM Integrated
11/20	Assembla site set up
11/27	SVN Trunk created at Assembla
12/12	Grammar Complete
12/15	Early test cases written
12/15	Tree walker draft done
12/15	Back-End draft done
12/15	LRM finalized
12/17	Browser Set up
12/18	Report Complete

11. Architectural Design
11.1 Block Diagram


Webpage
.html
Browser
.java
Masc
.java
MascAntlr Lexer
.java
grammar
.g
MascAntlr Parser
.java
walker
.g
MascAntlr Walker
.java
Masc Backend Files
.java

11.2 Interfaces

The general architecture of the MASC compiler is shown in the schematic diagram in Fig. 1. The web page is parsed by the HTML parser and this is fed as input to the MASC Browser (Browser.java). Browser.java in turn connects to the MascAntlrLexer which is generated from the Antlr grammar in the grammar.g file. The browser directs the code written by the programmer in the MASC shell (which is embedded in the browser), to the lexer. This is where the front end of the MASC compiler begins. The MascAntlrLexer takes the code sent by the lexer, scans it and forwards its output to MascAntlrParser, which is also generated from the Antlr grammar in grammar.g. The parser essentially takes the tokens generated by the lexer and forms an abstract syntax tree out of them. This AST is passed on to MascAntlrWalker, which is generated from the walker.g.

The walker now walks this AST and identifies the various sub-trees in the AST. According to the operations indicated at the respective roots of these sub-trees, the walker calls the corresponding functions in MascInterpreter.java in the compiler backend. The interpreter coordinates the various structures and functionalities available in the backend to generate code or pass on control to Java or the shell, as appropriate. Once compilation concludes, the program executes and the results are send back to the MASC shell in the browser.

11.3 Who did what

Kristina	Browser.java, grammar.g
Vaibhav	Walker.g
Christos	MASC Back-end

12. Test Plan

As you can probably tell form our presentation, there really wasn't much, if any time for testing. Here's what we did.

12.2 Test suites used

We used one test suite consisting of 7 examples, plus Christos and Vaibhav did some manual test of things while trying to debug their code.

12.3 Choosing Test Cases

Looking at what other groups had done in the past, we found two important points. First, it's important to test out the unique features of your language. For us, that meant making sure shell commands, browser interaction, and if/else syntax was good. We also put in some things with loops and expressions and arrays for good measure. Second, it's important to test the bad cases as well as the good. You don't want your interpreter accepting something it shouldn't.

12.4 Automation

Getting automated testing to work for us is extremely difficult, given that we had to worry about the browser and the shell commands, plus the fact that we switched from a Firefox plug-in to the MASC Broswer with Lobo at the last minute. Some of the things we tried were geared towards working as a plug-in. We also tried using RFT scripts and the shell script code that was given in class to try to make it work to no avail.

12.5 Who did what

Kristina	Test cases, shell scripting
Vaibhav	RFT scripting
Michael	Test cases,

13. Lessons Learned

13.1 Michael

I learned not to try programming in Windows. My new laptop ran Vista, and that was a pain to get anything working on…even dual booting with Fedora 7 (and 8) didn't help. It put me at a major disadvantage in terms of doing programming for all of my courses and especially this project, given that I feel I'm a weak programmer to begin with. But I also learned (perhaps too late), that I can't let my fear of failing and intimidation of my skills get in the way of helping out. I tried to help out wherever I could, and pushed myself to do things and try things I hadn't before. While you'll find my name is on hardly any of the code submitted, I went through all of it and made sure I understood what was going on. And yet, I feel that an extra set of hands in the programming might have been enough to handle this project. Even if we bit off more than we could chew.

13.3 Vaibahv

1) It is extremely important to remove any non-determinism from the grammar, to ensure smooth working of the walker.
2) The interfaces that the walker calls upon, at the backend of the compiler must be cleanly defined.
3) Nested calls to the functions in the interpreter by the walker must be resolved correctly.
4) The walker should try to clearly cast the arguments with which it calls the interpreter functions, but this is often difficult to implement and consequently, the backend should account for it.

13.4 Kristina

I learned:

-That Firefox has a surprisingly difficult-to-use extension interface. There is almost no documentation on how to do extensions using a Java, and what documentation there was often conflicted itself. I learned that Firefox is dumb in that you have to explicitly set a security policy to allow yourself to access files on the file system. If you can set the policy, though, you obviously have the rights to access files. I learned that it is probably easier to learn C++ and write your extension in that (which is more native to Firefox) than bother with Java. Most preferable would be doing the whole thing in JavaScript.

-That most people won't do anything unless someone is standing behind them with a stick and prodding.

-That people will take credit for your work, standing right next to you. I started writing that damn browser on Sunday (all by myself) and connected it to the interpreter last night. The team was not hampered by a lack of browser, as the shell can also be run without a broswer at the command line (type "java Masc"). Mike and I were the only ones to ever spend any time on the browser stuff.

-To avoid teamwork at all costs in the future. I could have finished this interpreter in a month working on my own. I should have just done that.

Appendix: Code listing

Browser

```
/*
 * Written by Kristina
 */

import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.net.*;

import org.lobobrowser.gui.*;
import org.lobobrowser.main.*;

import org.htmlparser.Parser;
import org.htmlparser.Node;
import org.htmlparser.NodeFilter;
import org.htmlparser.util.*;
import org.htmlparser.util.ParserException;
import org.htmlparser.parserapplications.*;
import org.htmlparser.tags.*;
import org.htmlparser.filters.*;

public class Browser extends JFrame implements KeyListener {

    // Global variables

    // Browser size
    static final int WIDTH = 1000;
    static final int HEIGHT = 700;
    // Browser pane
    final int B_WIDTH = WIDTH - 12;
    final int B_HEIGHT = HEIGHT - 110;
    // Command line
    final int CMD_WIDTH = WIDTH - 12;
    final int CMD_HEIGHT = HEIGHT - 600;


    // GUI
    static JTextArea cmdLine;
    static FramePanel framePanel;
```

```
// Navigation variables
static String HOME;
static String currentURL;

// Command line
static String PS1;

Masc shell;
public final int SHELL_ROWS = 6;

public static void getGlobalVars() {
	PS1 = ">>> ";
	HOME = "http://www.google.com";
}

public static void main(String[] args) throws Exception {
	// This step is necessary for extensions to work:
	PlatformInit.getInstance().init(args, true);

	getGlobalVars();

	// Create frame with a specific size.
	JFrame frame = new Browser();
	frame.addWindowListener(new MyWindowListener());
	frame.setSize(WIDTH,HEIGHT);
	frame.setTitle("MASC Browser");
	frame.setVisible(true);

	framePanel = (FramePanel)frame.getContentPane().getComponent(0);
	cmdLine = (JTextArea)frame.getContentPane().getComponent(1);
	cmdLine.setText(PS1);
	cmdLine.setCaretPosition(PS1.length());
}

ByteArrayOutputStream byteStream;
public PrintStream stdout;

ByteArrayOutputStream errStream;
public PrintStream stderr;

public final int OUTPUT_SIZE=100;
LinkedList output = new LinkedList<String>();


public Browser() throws Exception {
	Container c = this.getContentPane();
```

```
        FramePanel framePanel = new FramePanel();
        framePanel.setPreferredSize(new Dimension(B_WIDTH, B_HEIGHT));
        framePanel.setMinimumSize(new Dimension(B_WIDTH, B_HEIGHT));

        JTextArea cmdLine = new JTextArea();
        cmdLine.addKeyListener(this);
        cmdLine.setPreferredSize(new Dimension(CMD_WIDTH, CMD_HEIGHT));
        cmdLine.setMaximumSize(new Dimension(CMD_WIDTH, CMD_HEIGHT));
        cmdLine.setFont(new Font("monospaced", Font.PLAIN, 12));
        cmdLine.setLineWrap(true);
        output.add(PS1);

        c.setLayout(new BoxLayout(c, BoxLayout.Y_AXIS));
        c.add(framePanel);
        c.add(cmdLine);
        framePanel.navigate("http://google.com");

        shell = new Masc();
        shell.init_Masc();
        shell.verbose = false;
        byteStream = new ByteArrayOutputStream();
        stdout = new PrintStream(byteStream);
        System.setOut(stdout);

        errStream = new ByteArrayOutputStream();
        stderr = new PrintStream(errStream);
        System.setErr(stderr);
    }


    // handling key events

    private final int NEWLINE = 10;
    private final String MASC_SUFFIX = ".ms";

    public void keyTyped(KeyEvent k) {
        int code = k.getKeyChar();
        if(code == NEWLINE) {
            output.removeLast();
            // get the last command entered
            output.add(getLastLine());
            String cmd = ((String)output.getLast()).substring(PS1.length()).trim();

            // add cmd to history
            addToHistory(cmd);
```

```
            // hack!  hack!  hack!
            // move to simple expression evaluation in interpreter?
            if(cmd.endsWith(MASC_SUFFIX)) {
                  try {
                     Scanner sc = new Scanner(new File(cmd));
                     cmd = "";
                     while(sc.hasNextLine())
                           cmd += sc.nextLine();
                  }
                  catch (IOException e) {
                     //then just try to run the cmd as is
                  }
            }
            cmd+="\n";

            String source = framePanel.getSourceCode();
            shell.execCmd((InputStream)new ByteArrayInputStream(cmd.getBytes()));

            // display new cmd prompt
            handleOutput();
            cmdLine.setText(getCmdPrompt());
         }
   }

   public String getLastLine() {
         StringTokenizer toke = new StringTokenizer(cmdLine.getText(), "\n");
         int numLines = toke.countTokens();
         for(int i=0; i<numLines-1; i++)
            toke.nextToken();

         return toke.nextToken()+"\n";
   }

   public void handleOutput() {
         StringTokenizer toke = new
StringTokenizer(byteStream.toString()+errStream.toString(), "\n");
         int numLines = toke.countTokens();
         String s = "";
         while(output.size() < OUTPUT_SIZE && numLines > 0) {
            output.add(toke.nextToken()+"\n");
            numLines--;
         }

         while(numLines > 0) {
            output.add(toke.nextToken()+"\n");
```

```
            output.remove();
            numLines--;
        }
        byteStream.reset();
        errStream.reset();
}

public int startRow;
public int endRow;

public String getCmdPrompt() {
        output.add(PS1);
        startRow = output.size()-(Math.min(SHELL_ROWS, output.size()));
        endRow = output.size();

        return getCmdPrompt(startRow, endRow);
}

public String getCmdPrompt(int start, int end) {
        String s = "";
        for(int i=start; i<end; i++) {
            s += output.get(i);
        }
        return s;
}

public void alert(String s) {
        JOptionPane.showMessageDialog(null, s);
}

public void keyPressed(KeyEvent k) {
        if(k.isShiftDown() && k.getKeyCode()==KeyEvent.VK_UP) {
            if(startRow > 0) {
                startRow--;
                endRow--;

                cmdLine.setText(getCmdPrompt(startRow, endRow));
            }
        }
        else if(k.isShiftDown() && k.getKeyCode()==KeyEvent.VK_DOWN) {
            if(startRow < output.size()-SHELL_ROWS) {
                startRow++;
                endRow++;

                cmdLine.setText(getCmdPrompt(startRow, endRow));
            }
```

```
        }

}
public void keyReleased(KeyEvent k) {    }


public int HISTORY_SIZE = 20;
public LinkedList history;

public void addToHistory(String cmd) {
        if(history == null) {
            history = new LinkedList<String>();
        }

        if(history.size() == HISTORY_SIZE) {
            history.remove();
        }
        history.add(cmd);
}

/*    public String printHistory() {
        String h = "";
        ListIterator hi = history.listIterator();
        int i=1;
        while(hi.hasNext())
            h += (i++)+"\t"+hi.next()+"\n";
        return h;
        }*/



// MASC library functions
public static void go(DataType d) {
        try {
            currentURL = d.getName();
            framePanel.navigate(d.getName());
        }
        catch(MalformedURLException e) {
            System.out.println("Bad URL");
            e.printStackTrace();
        }
}
public static void reload() {
            framePanel.reload();
}
```

```
    public static boolean back() {
            return framePanel.back();
    }

    public static boolean forward() {
            return framePanel.forward();
    }
    public static void exit() {
          System.exit(0);
    }



    //HTML Parser functions
    public static void extractLink(String regex) {
       NodeList list = new NodeList();

       try {
          Parser parser = new Parser(currentURL);
          NodeFilter filter = new LinkRegexFilter(regex);

          for (NodeIterator i = parser.elements(); i.hasMoreNodes(); ) {
             i.nextNode().collectInto(list, filter);
          }
       }
       catch (ParserException e) {
          System.out.println("Parser exception error.");
          System.out.println(e.getStackTrace());
       }

       for(int i=0; i<list.size(); i++) {
          System.out.println(i+"\t"+(list.elementAt(i)).toPlainTextString());
       }
    }


}

class MyWindowListener implements WindowListener {
    public void windowClosing(WindowEvent arg0) {
          System.exit(0);
    }

    public void windowOpened(WindowEvent arg0) {}
    public void windowClosed(WindowEvent arg0) {}
    public void windowIconified(WindowEvent arg0) {}
```

```
    public void windowDeiconified(WindowEvent arg0) {}
    public void windowActivated(WindowEvent arg0) {}
    public void windowDeactivated(WindowEvent arg0) {}
}
```

Build

```
#!/bin/bash

#
# Written by Kristina
#

#
# Compiles and runs everything
#

rm *.class
echo "Building from scratch..."

echo -n "Setting CLASSPATH..."
CLASSPATH=$CLASSPATH:./lobo.jar:./htmlparser.jar
export CLASSPATH
echo "done"

echo -n "Generating lexer/parser..."
antlr grammar.g
echo "done"

echo -n "Generating walker..."
antlr walker.g
echo "done"

echo -n "Compiling..."
javac *.java
echo "done"

java Browser
```

darSh

```
import java.io.*;

/***
 *
 * @author Christos Angelopoulos
 *
 */
```

```
public class darSh {
        static InputStreamReader in;
   static StringBuffer buff;

   public static void main(String[] args) {
      try {
         String os = System.getProperty("os.name");
         String shell = "sh";
         String c = "-c";
         if(os.contains("Mac") || os.contains("Unix") || os.contains("Linux")) {
                  shell = "sh";
                  c = "-c";
         }
         else if(os.contains("Win")) {
                  shell = "cmd";
                  c = "/c";
         }

         in = new InputStreamReader(System.in);
         String cmd = "";
         while(true) {
            printPrompt();
            if((cmd = getCmd()).compareTo("exit") == 0) {
                  break;
            }

            String[] command =  { shell, c, cmd };
            final Process process = Runtime.getRuntime().exec(command);
            new Thread()
            {
               public void run()
               {
                  try{
                     InputStream is = process.getInputStream();
                     byte[] buffer = new byte[1024];
                     for(int count = 0; (count = is.read(buffer)) >= 0;)
                     {
                        System.out.write(buffer, 0, count);
                     }
                  }
                  catch(Exception e)
                  {
                     e.printStackTrace();
                  }
               }
            }.start();
```

```
            new Thread()
            {
               public void run()
               {
                  try{
                     InputStream is = process.getErrorStream();
                     byte[] buffer = new byte[1024];
                     for(int count = 0; (count = is.read(buffer)) >= 0;)
                     {
                        System.err.write(buffer, 0, count);
                     }
                  }
                  catch(Exception e)
                  {
                     e.printStackTrace();
                  }
               }
            }.start();

            int returnCode = process.waitFor();
            System.out.println("Return code = " + returnCode);

         }
         in.close();


      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
}

         public static void printPrompt() {
                  System.out.print(" >>> ");
         }

         public static String getCmd() {
                  try {
                     buff = new StringBuffer("");
                     char ch;
                     while((ch = (char)in.read()) != '\n') {
                           buff.append(ch);
                     }
                  }
```

```
                catch (IOException e) {
                   System.out.println("Couldn't read input!");
                }
                return buff.toString();
        }
}
```

Data Type

```
import java.io.PrintWriter;

/***
 *
 * @author Christos Angelopoulos
 * Kristina - added toString() and getTypeStr()
 *
 */

public class DataType {

   private String name = "";

   public DataType() {
        name = "";
   }

   public DataType(String s) {
        name = s;
   }

   public static final int intType = 0;
   public static final int floatType = 1;
   public static final int stringType = 2;
   public static final int boolType = 3;
   public static final int funcType = 4;

   public String toString() {
        return this.name+"("+this.getTypeStr()+")";
   }

   public String getTypeStr() {
        if(this instanceof MascInt) {
           return "int";
        }
        else if(this instanceof MascFloat) {
           return "float";
```

```
        }
        else if(this instanceof MascString) {
            return "str";
        }
        else if(this instanceof MascBool) {
            return "bool";
        }
        else if(this instanceof MascFunction) {
            return "function";
        }
        return "other";
    }

        public static int getType(DataType t) {
                if(t instanceof MascInt) {
                        return intType;
                }
                else if(t instanceof MascFloat) {
                        return floatType;
                }
                else if(t instanceof MascString) {
                        return stringType;
                }
                else if(t instanceof MascBool) {
                        return boolType;
                }
                else if(t instanceof MascFunction) {
                    return funcType;
                }
                return -1;
        }

        public static final DataType
                mInt = new DataType("int"),
                mFloat = new DataType("float"),
                mString = new DataType("string"),
                mBool = new DataType("bool");

        public static boolean numeric(DataType p) {
                return p == DataType.mInt || p == DataType.mFloat;
        }

        public static DataType max(DataType p1, DataType p2) {
                if (!numeric(p1) || !numeric(p2)) {
                        return null;
                }
```

```
                else if (p1 == DataType.mFloat || p2 == DataType.mFloat) {
                        return DataType.mFloat;
                }
                else {
                        return DataType.mInt;
                }
        }

        public DataType error(String mesg) {
                throw new MascException(mesg + " " + getName());
        }

        public DataType error(DataType dt, String mesg) {
    if (dt == null) {
        return error(mesg);
    }
    throw new MascException(mesg + " " + getName());
}

        /**
         * @return the name
         */
        public String getName() {
                return name;
        }

        /**
         * @param name the name to set
         */
        public void setName(String name) {
                this.name = name;
        }

        public void print(PrintWriter w) {
    if(name != null) {
        w.print( name + " = " );
    }
}

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

        /* (non-Javadoc)
         * @see java.lang.Object#hashCode()
         */
```

```
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}

	/* (non-Javadoc)
	 * @see java.lang.Object#equals(java.lang.Object)
	 */
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (!(obj instanceof DataType))
			return false;
		final DataType other = (DataType) obj;
		if (name == null) {
			if (other.name != null)
				return false;
		} else if (!name.equals(other.name))
			return false;
		return true;
	}

	public DataType assign(DataType dt) {
   return error(dt, "=");
}

public DataType uminus() {
   return error("-");
}

public DataType plus(DataType dt) {
		return error(dt, "+");
}

public DataType add(DataType dt) {
   return error(dt, "+=");
}

public DataType minus(DataType dt) {
   return error(dt, "-");
```

```
}

public DataType sub(DataType dt) {
   return error(dt, "-=");
}

public DataType multiply(DataType dt) {
   return error(dt, "*");
}

public DataType mul(DataType dt) {
   return error(dt, "*=");
}

public DataType divide(DataType dt) {
   return error(dt, "/");
}

public DataType div(DataType dt) {
   return error(dt, "/=");
}

public DataType modulus(DataType dt) {
   return error(dt, "%");
}

public DataType remainder(DataType dt) {
   return error(dt, "%=");
}

public DataType gt(DataType dt) {
   return error(dt, ">");
}

public DataType ge(DataType dt) {
   return error(dt, ">=");
}

public DataType lt(DataType dt) {
   return error(dt, "<" );
}

public DataType le(DataType dt) {
   return error(dt, "<=");
}
```

```
    public DataType eq(DataType dt) {
        return error(dt, "==");
    }

    public DataType ne(DataType dt) {
        return error(dt, "!=");
    }

    public DataType and(DataType dt) {
        return error(dt, "and");
    }

    public DataType nand(DataType dt) {
        return error(dt, "nand");
    }

    public DataType or(DataType dt) {
        return error(dt, "or");
    }

    public DataType xor(DataType dt) {
        return error(dt, "xor");
    }

    public DataType xnor(DataType dt) {
        return error(dt, "xnor");
    }

    public DataType not() {
        return error("not");
    }

}
```

Driver

```
package env;

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

public class Driver {

    static InputStreamReader in;
    static StringBuffer buff;

    public static void main(String[] args) throws IOException {
            in = new InputStreamReader(System.in);
```

```
        String cmd = "";
        while(true) {
           printPrompt();
           if((cmd = getCmd()).compareTo("exit") == 0) break;
           List ps = cmdToList(cmd);
           run(ps);
        }
        in.close();
}

public static void printPrompt() {
           System.out.print(" >>> ");
}

public static String getCmd() {
        try {
           buff = new StringBuffer("");
           char ch;
           while((ch = (char)in.read()) != '\n') {
                 buff.append(ch);
           }
        }
        catch (IOException e) {
           System.out.println("Couldn't read input!");
        }

        return buff.toString();
}

// Make a list out of "cmd arg1 arg2 ... "
public static List cmdToList(String s) {
        StringTokenizer st = new StringTokenizer(s);
        List ps = new ArrayList();
        while(st.hasMoreTokens()) {
           ps.add(st.nextToken());
        }
        return ps;
}

// run the ps
public static void run(List ps) {
        // standard input reader
        Scanner sc = new Scanner(System.in);
        char in;

        ProcessBuilder pb = new ProcessBuilder(ps);
```

```
        pb.redirectErrorStream(true);
        try {
           char ch;
           Process x = pb.start();
           while((ch = (char)(x.getInputStream().read())) != 0xFFFF) {
                 System.out.print(ch);
                 while(sc.hasNextByte()) {
                    in = (char)sc.nextByte();
                    System.out.println(in);
                    x.getOutputStream().write(in);
                 }
           }
        }
        catch(IOException e) {
           System.out.println("unable to start ps: "+e.getStackTrace());
        }
   }
}
```

Grammar.g

```
/* Written by Kristina */

class MascAntlrLexer extends Lexer;

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



{
   private static boolean done = false;

   public void uponEOF()
      throws TokenStreamException, CharStreamException
   {
      done=true;
   }

   public static void main(String[] args) throws Exception {
      MascAntlrLexer lexer = new MascAntlrLexer(System.in);
      while ( !done ) {
         Token t = lexer.nextToken();
         System.out.println("Token: "+t);
```

```
      }
   }

}

protected ALPHA : 'a'..'z' | 'A'..'Z' | '_';
protected DIGIT : '0'..'9';

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

NL : ('\n' | ('\r' '\n') => '\r' '\n' | '\r') { newline(); } ;

COMMENT : "(:" ( options { greedy = false; } : .)* ":)"
      { $setType(Token.SKIP); } ;

// parens and braces
LPAREN : '(';
RPAREN : ')';
LBRACE : '{';
RBRACE : '}';

// arithmetic ops
PLUS : '+';
MINUS : '-';
STAR : '*';
DIV : '\\';

// logical ops
NOT : '!';
AND : "&&";
OR : "||";

// comparison ops
LT : '<';
GT : '>';
EQ : "==";
LTE : "<=";
GTE : ">=";
NE : "!=";

// misc
ASG : '=';
SEMI : ';';
COMMA : ',';

// redirection
```

```
REDIR_OUT : "->";
REDIR_IN : "<-";
REDIR_APP : ">>";

// piping
PIPE : '|';

// background
BG : '&';

ID options { testLiterals = true; } : ( (':' | '~')? '/' | ALPHA | DIGIT | '.' )+;

// array syntax
LSBRACE : '[';
RSBRACE : ']';


// As filenames can be numbers, the lexer can't distinguish between the two
//NUMBER : (DIGIT)+ (('.' (DIGIT)+ { $setType(FLOAT); } ) | {
$setType(INTEGER); }) ;


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


// Parser
class MascAntlrParser extends Parser;

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

tokens {
   //PROGRAM;
   CMD;
   ARRAY;
   EXPR_LIST;
   FUNC_CALL;
   UMINUS;
   VAR_LIST;
}


{
```

```
   public void uponEOF() throws Exception
   {
      System.out.println("EOF");
   }

   public static void main(String[] args) throws Exception {
      MascAntlrLexer lexer = new MascAntlrLexer(System.in);
      MascAntlrParser parser = new MascAntlrParser(lexer);
      // Parse the input expression
      parser.program();
      AST t = parser.getAST();
      if(t == null) {
         System.out.println("AST is null!");
      }
      else {
         printTree(t, "");
      }
   }

   public static void printTree(AST t, String tabs) {
      int x = t.getNumberOfChildren();
      System.out.println(tabs+t.toString()+": "+x);
      if(x > 0) {
         printTree(t.getFirstChild(), tabs+"\t");
      }

      if(t.getNextSibling() != null) {
         printTree(t.getNextSibling(), tabs);
      }
   }
}


// the entire program
program : ((cmd)? (NL! | SEMI!))* EOF!
      {#program = #([CMD,"PROGRAM"], program); }
   ;

assignment_word
   : l_value ASG^ expr
   | ID ( LSBRACE! expr RSBRACE!
        {#assignment_word = #([ARRAY,"ARRAY"], assignment_word); })+ ASG!
expr
   ;
```

```
// redirection
redirection
   : (REDIR_IN^ | REDIR_OUT^ | REDIR_APP^) ID;

simple_cmd
   : (ID^ (l_value)* (PIPE^ ID^ (l_value)*)*) (redirection)*;

cmd
   : simple_cmd
   | assignment_word
   | control
   | while_cmd
   | if_cmd
   | func_def
   | array
   | func_call
      {#cmd = #([CMD,"CMD"], cmd); }
   ;

cmd_list
   : ((cmd)? (NL! | SEMI!))*;

// control flow
while_cmd
   : "while"^
      LPAREN! expr RPAREN! (NL!)*
      LBRACE! cmd_list RBRACE!;

if_cmd
   : "if"^ LPAREN! expr RPAREN! (NL!)* LBRACE! cmd_list
         ("else"! cmd_list)? RBRACE!
   ;

func_def
   : "function"^ ID LPAREN! var_list  RPAREN! (NL!)*
      LBRACE! cmd_list RBRACE!;

var_list
      : ID ( COMMA! ID )*
         {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
      |  /* nothing */
         {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
      ;

// control flow shortcuts
control
```

```
   : ("return"^ (expr)?) | "continue"^ | "break"^;

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

expr_list
   : expr (COMMA! expr)*
      {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
   | /* empty list*/
      {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
   ;


expr
   : logic_factor ( (AND^ | OR^) logic_factor )* ;

logic_factor
   : (NOT^)? relat_expr;

relat_expr
   : arith_expr ( (GTE^ | LTE^ | GT^ | LT^ | EQ^ | NE^) arith_expr )?;

arith_expr
   : arith_term ( (PLUS^ | MINUS^) arith_term )*;

arith_term
   : arith_factor ( (STAR^ | DIV^ ) arith_factor )*;

arith_factor
   : MINUS! r_value
      {#arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
   | r_value;

r_value
   : l_value
   | func_call
   | STRING
   | "TRUE"
   | "FALSE"
   | (LPAREN! expr RPAREN!)
   ;

l_value
      : ID^
```

```
    ;

array
    : "array"^ ID ( LSBRACE! expr RSBRACE!)+
    ;
```

Masc

```
import javax.swing.*;

import org.htmlparser.parserapplications.StringExtractor;
import org.htmlparser.util.ParserException;

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

/***
 *
 * @author Christos Angelopoulos
 * Kristina - wrote init_Masc and execCmd (which is modified from execFile)
 * and made the walker global so that the symbol table would persist from
 * command to command in the shell (the walker, when instantiated, creates
 * the symbol table).
 *
 */

public class Masc {

   public static boolean verbose = true;


   MascAntlrLexer lexer;
   MascAntlrParser parser;

   MascAntlrWalker walker;
   public void init_Masc() {
            walker = new MascAntlrWalker();
   }

   public String execCmd(InputStream cmd) {
```

```
        lexer = new MascAntlrLexer(cmd);
        parser = new MascAntlrParser(lexer);
        try {
           parser.program();
           CommonAST tree = (CommonAST)parser.getAST();

           // Print the resulting tree out in LISP notation
           if ( verbose ) {
                 System.out.println(
                                 "================ tree structure
======================" );
                 System.out.println( tree.toStringList() );
           }


           // Traverse the tree created by the parser

           if ( verbose )
                 System.out.println(
                              "================ program output
======================" );

           DataType r = walker.expr(tree);
           if ( verbose )
                 System.out.println(
                                 "================ program return
======================" );
        }
        catch(Exception e) {
           e.printStackTrace();
           return "error!";
        }


        return "";
   }



        public static void execFile(String filename) {
      try {
        InputStream input = (filename != null) ? (InputStream) new FileInputStream(
filename ) : (InputStream) System.in;

        MascAntlrLexer lexer = new MascAntlrLexer(input);
```

```
        MascAntlrParser parser = new MascAntlrParser(lexer);

        // Parse the input program
        parser.program();

        CommonAST tree = (CommonAST)parser.getAST();
        if ( verbose )
        {
           // Print the resulting tree out in LISP notation
           System.out.println(
              "================ tree structure ======================" );
           System.out.println( tree.toStringList() );
        }


        MascAntlrWalker walker = new MascAntlrWalker();
        // Traverse the tree created by the parser

        if ( verbose )
           System.out.println(
              "================ program output ======================" );

        DataType r = walker.expr(tree);

        if ( verbose )
           System.out.println(
              "================ program return ======================" );
        if ( null != r )
           r.print();



    } catch(IOException e) {
       System.err.println("Error: I/O: " + e);
    } catch(RecognitionException e) {
       System.err.println("Error: Recognition: " + e);
    } catch(TokenStreamException e) {
       System.err.println("Error: Token stream: " + e);
    } catch(Exception e) {
       System.err.println("Error: " + e);
    }
}

       public static void commandLine() {
    InputStream input = (InputStream) new DataInputStream(System.in);
    MascAntlrWalker walker = new MascAntlrWalker();
```

```
    while(true)  {
       try {
          while(input.available() > 0)
             input.read();
       }
       catch (IOException e) {
                System.err.println("Error in reading input");
       }

       System.out.print( "Masc:$ " );
       System.out.flush();

       MascAntlrLexer lexer = new MascAntlrLexer(input);
       MascAntlrParser parser = new MascAntlrParser(lexer);

       try {
          //parser.cmd_list();
          parser.cmd();
          CommonAST tree = (CommonAST)parser.getAST();
          DataType r = walker.expr(tree);
          if (r != null) {
             r.print();
          }
       } catch(RecognitionException e) {
          System.err.println("Recognition exception: " + e);
       } catch(TokenStreamException e) {
          if (e instanceof TokenStreamIOException) {
             System.err.println("Token I/O exception");
             break;
          }
          System.err.println("Error: Token stream: " + e);
       } catch(Exception e) {
          System.err.println("Error: Interpretive: " + e);
          e.printStackTrace();
       }
    }
}

       public static void main(String[] args) {
                //verbose = args.length >= 1 && args[0].equals( "-v" );

    boolean batch = args.length >=1 && args[0].equals( "-b" );

    if (args.length >= 1 && args[args.length - 1].charAt(0) != '-' ) {
       execFile( args[args.length-1] );
```

```
        }
        else if (batch) {
           execFile(null);
        }
        else {
           commandLine();
        }

        System.exit( 0 );
           }
}
```

MascArray

```
/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascArray extends DataType {
        private DataType t;
        private int size = 1;
        private DataType[] mArray;

        public MascArray(int sz, DataType tp) {
                super("[]");
                size = sz;
                t = tp;
                if(size < 0) {
                        throw new MascException("Array size " + size + " must be non
negative");
                }
                else {
                        mArray = new DataType[size];
                }
        }

        /* (non-Javadoc)
         * @see java.lang.Object#toString()
         */
        @Override
        public String toString() {
                // TODO Auto-generated method stub
                return "[ " + size + " ] " + t.toString();
        }

        public boolean checkIndex(int index) {
```

```
		if((index >= 0) && (index < size)) {
			return true;
		}
		return false;
	}

	public DataType get(int index) {
		if(checkIndex(index)) {
			return mArray[index];
		}
		throw new MascException("index out of bounds: " + index + " size: " +
size);
	}

	public void set(int index, DataType dt) {
		if(!checkIndex(index)) {
			throw new MascException("index out of bounds: " + index + "
size: " + size);
		}
		if(t instanceof MascBool) {
			if(dt instanceof MascBool) {
				mArray[index] = dt;
			}
		}
		if(t instanceof MascInt) {
			if(dt instanceof MascInt) {
				mArray[index] = dt;
			}
			MascInt mint;
			try {
				mint = new MascInt(MascInt.intValue(dt));
				mArray[index] = (DataType)mint; //MascInt.intValue(dt);
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		if(t instanceof MascFloat) {
			if(dt instanceof MascFloat) {
				mArray[index] = dt;
			}
			MascFloat mfloat;
			try {
				mfloat = new MascFloat(MascFloat.floatValue(dt));
				mArray[index] = (DataType)mfloat;
//MascInt.intValue(dt);
```

```
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                if(t instanceof MascString) {
                        if(dt instanceof MascString) {
                                mArray[index] = dt;
                        }
                        if(dt instanceof MascInt) {
                                Integer mint;
                                try {
                                        mint = MascInt.intValue(dt);
                                        MascString mstring = new
MascString(mint.toString());
                                        mArray[index] = (DataType)mstring;
//MascInt.intValue(dt);
                                } catch (Exception e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                        if(dt instanceof MascFloat) {
                                Float mfloat;
                                try {
                                        mfloat = MascFloat.floatValue(dt);
                                        MascString mstring = new
MascString(mfloat.toString());

                                        mArray[index] = (DataType)mstring;
//MascInt.intValue(dt);
                                } catch (Exception e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                }
        }

        /**
         * @return the t
         */
        public DataType getT() {
                return t;
        }
```

```
	/**
	 * @param t the t to set
	 */
	public void setT(DataType t) {
		this.t = t;
	}

	/**
	 * @return the size
	 */
	public int getSize() {
		return size;
	}

	/**
	 * @param size the size to set
	 */
	public void setSize(int size) {
		this.size = size;
	}

	/**
	 * @return the mArray
	 */
	public DataType[] getMArray() {
		return mArray;
	}

	/**
	 * @param array the mArray to set
	 */
	public void setMArray(DataType[] array) {
		mArray = array;
	}
}
```

MascBool

```
/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascBool extends DataType {

	private static boolean TRUE = true;
	private static boolean FALSE = false;
```

```
private boolean var;

public MascBool(boolean v) {
	//super("bool");
	// TODO Auto-generated constructor stub
	var = v;
}

 public String getType() {
	 return "bool";
 }

 public DataType and(DataType dt) {
	 if (dt instanceof MascBool) {
		 return new MascBool(var && ((MascBool)dt).var);
	 }
	 return error(dt, "and");
 }

 public DataType or(DataType dt) {
	 if (dt instanceof MascBool) {
		 return new MascBool(var || ((MascBool)dt).var);
	 }
	 return error(dt, "or");
 }

 public DataType xor(DataType dt) {
	 if (dt instanceof MascBool) {
		 boolean t = ((MascBool)dt).var;
		 return new MascBool((var && t) || (!var && !t));
	 }
	 return error( dt, "!=" );
 }

 public DataType xnor(DataType dt) {
	 if (dt instanceof MascBool) {
		 boolean t = ((MascBool)dt).var;
		 return new MascBool((!var && t) || (var && !t));
	 }
	 return error( dt, "!=" );
 }

 public DataType nand(DataType dt) {
	 if (dt instanceof MascBool) {
		 return new MascBool(!(var && ((MascBool)dt).var));
```

```
                }
                return error(dt, "and");
        }

        public DataType not() {
                return new MascBool(!var);
        }

        public DataType eq(DataType dt) {
                if (dt instanceof MascBool) {
                        return new MascBool((var && ((MascBool)dt).var) || (!var &&
!((MascBool)dt).var));
                }
            return error(dt, "==");
          }

        public DataType ne(DataType dt) {
                if (dt instanceof MascBool) {
                        return new MascBool((var && !((MascBool)dt).var) || (!var &&
((MascBool)dt).var));
                }
                return error( dt, "!=" );
        }


       /**
        * @return the var
        */
       public boolean getVar() {
               return var;
       }

       /**
        * @param var the var to set
        */
       public void setVar(boolean var) {
               this.var = var;
       }

       /**
        * @return the tRUE
        */
       public static boolean getTRUE() {
               return TRUE;
       }
```

```
	/**
	 * @return the fALSE
	 */
	public static boolean getFALSE() {
		return FALSE;
	}

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

}
```

MascException

```
/***
 *
 * @author Christos Angelopoulos
 *
 */
public class MascException extends RuntimeException {
   MascException(String msg) {
      System.err.println( "Error: " + msg );
   }
}
```

MascFloat

```
import java.io.PrintWriter;

/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascFloat extends DataType {

	private float var;

	public MascFloat(float v) {
		//super("float");
		// TODO Auto-generated constructor stub

		var = v;
	}

	public static float floatValue(DataType dt) {
		if(dt instanceof MascFloat) {
			return ((MascFloat)dt).var;
```

```
		}
		if(dt instanceof MascInt) {
			return (float)((MascInt)dt).getVar();
		}
		try {
			Float a = Float.parseFloat(dt.getName());

			if(a instanceof Float) {
				return a;
			}
		}catch(Exception e) {
			System.err.println(e);
		}
		dt.error("problem casting to float");
		return 0;
	}

	public MascFloat uminus() {
		return new MascFloat(-var);
	}

	public DataType assign(DataType dt) {
		var = MascFloat.floatValue(dt);
		float b = var;
		DataType mfloat = new MascFloat(b);

		mfloat.setName(Float.toString(b));
		return mfloat;
		//return new MascFloat(var = floatValue(dt));
	}

	public DataType plus(DataType dt) {
		float b = var + MascFloat.floatValue(dt);
		DataType mfloat = new MascFloat(b);

		mfloat.setName(Float.toString(b));
		return mfloat;
		//return new MascFloat(var + MascFloat.floatValue(dt));
	}

	public DataType add(DataType dt) {
		var += floatValue(dt);
		return this;
	}

	public DataType minus(DataType dt) {
```

```
        float b = var - MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
        return mfloat;
        //return new MascFloat(var - MascFloat.floatValue(dt));
}

public DataType sub(DataType dt) {
        var -= floatValue(dt);
        return this;
}

public DataType multiply(DataType dt) {
        float b = var * MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
        return mfloat;
        //return new MascFloat(var * MascFloat.floatValue(dt));
}

public DataType mul(DataType dt) {
        var *= floatValue(dt);
        return this;
}

public DataType divide(DataType dt) {
        float b = var / MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
        return mfloat;
        //return new MascFloat(var / MascFloat.floatValue(dt));
}

public DataType div(DataType dt) {
        var /= floatValue(dt);
        return this;
}

public DataType modulus(DataType dt) {
        float b = var % MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
```

```
            return mfloat;
            //return new MascFloat(var % MascFloat.floatValue(dt));
        }

public DataType remainder(DataType dt) {
   var %= floatValue(dt);
   return this;
}

        public DataType gt(DataType dt) {
            //return new MascFloat(var > MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var > floatValue(dt));
        }

        public DataType ge(DataType dt) {
            //return new MascFloat(var >= MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var >= floatValue(dt));
        }

        public DataType lt(DataType dt) {
            //return new MascFloat(var < MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var < floatValue(dt));
        }

        public DataType le(DataType dt) {
            //return new MascFloat(var <= MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var <= floatValue(dt));
        }

        public DataType eq(DataType dt) {
            //return new MascFloat(var == MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var == floatValue(dt));
        }

        public DataType ne(DataType dt) {
            //return new MascFloat(var != MascFloat.floatValue(dt) ? 1 : 0);
            return new MascBool(var != floatValue(dt));
        }

        /**
         * @return the var
         */
        public float getVar() {
            return var;
        }
```

```
	/**
	 * @param var the var to set
	 */
	public void setVar(float var) {
		this.var = var;
	}

	public void print(PrintWriter pw) {
		pw.println(((Float)var).toString());
	}

	public void print() {
		System.out.println(Float.toString(var));
	}
}
```

MascFunction

```
import antlr.collections.AST;

/***
 *
 * @author Christos Angelopoulo
 *
 */

public class MascFunction extends DataType {

	private String[] args;
	private AST body;
	private MascSymbolTable fst;
	private int id;

	public MascFunction(String function_name, String[] args, AST body,
MascSymbolTable st) {
		//super(function_name);
		// TODO Auto-generated constructor stub
		this.args = args;
		this.body = body;
		this.fst = st;
	}

	public MascFunction(String function_name, int id) {
		//super(function_name);
		this.args = null;
		this.id = id;
		fst = null;
	   body = null;
```

```
        }

public final boolean isLibrary() {
   return body == null;
}

        /**
         * @return the args
         */
        public String[] getArgs() {
                return args;
        }

        /**
         * @param args the args to set
         */
        public void setArgs(String[] args) {
                this.args = args;
        }

        /**
         * @return the body
         */
        public AST getBody() {
                return body;
        }

        /**
         * @param body the body to set
         */
        public void setBody(AST body) {
                this.body = body;
        }

        /**
         * @return the fst
         */
        public MascSymbolTable getSymbolTable() {
                return fst;
        }

        /**
         * @param fst the fst to set
         */
        public void setSymbolTable(MascSymbolTable fst) {
                this.fst = fst;
```

```
	}

	public String getTypeName() {
		return "function";
	}

	/**
	 * @return the id
	 */
	public int getId() {
		return id;
	}

	/**
	 * @param id the id to set
	 */
	public void setId(int id) {
		this.id = id;
	}
}
```

MascInt

```
import java.io.PrintWriter;

/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascInt extends DataType {

	private int var;

	public MascInt(int v) {
		//super("int");
		var = v;
		// TODO Auto-generated constructor stub
	}

	public String getTypeName() {
		return "int";
	}

	public static int intValue(DataType dt) {
		if(dt instanceof MascFloat) {
			return (int)((MascFloat)dt).getVar();
```

```
		}
		if(dt instanceof MascInt) {
			return ((MascInt)dt).var;
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {
				System.out.println("a: " + a);
				return a;
			}
		}catch(Exception e) {
			System.err.println(e);
		}
		dt.error("problem casting to int");
		return 0;
	}

	public MascInt uminus() {
		return new MascInt(-var);
	}

	public DataType assign(DataType dt) {
		var = MascInt.intValue(dt);
		int b = var;
		DataType mint = new MascInt(b);

		mint.setName(Integer.toString(b));
		return mint;
		//return new MascInt(var = intValue(dt));
	}

	public DataType plus(DataType dt) {
		if(dt instanceof MascInt) {
			int b = var + intValue(dt);
			DataType mint = new MascInt(b);
			mint.setName(Integer.toString(b));
			return mint;
			//return new MascInt(var + intValue(dt));
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {

				int b = var + a;
```

```
                        DataType mint = new MascInt(b);
                        mint.setName(Integer.toString(b));
                        return mint;
                        //return new MascInt(var + a);
                }
        }catch(Exception e) {
                System.err.println(e);
        }
        float b = var + MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
        return mfloat;
        //return new MascFloat(var + MascFloat.floatValue(dt));
}

public DataType add(DataType dt) {
        var += intValue(dt);
        return this;
}

public DataType minus(DataType dt) {
        /*if(dt instanceof MascInt) {
                return new MascInt(var - intValue(dt));
        }
        return new MascFloat(var - MascFloat.floatValue(dt));*/

        if(dt instanceof MascInt) {
                int b = var - intValue(dt);
                DataType mint = new MascInt(b);
                mint.setName(Integer.toString(b));
                return mint;
                //return new MascInt(var + intValue(dt));
        }
        try {
                Integer a = Integer.parseInt(dt.getName());

                if(a instanceof Integer) {

                        int b = var - a;
                        DataType mint = new MascInt(b);
                        mint.setName(Integer.toString(b));
                        return mint;
                        //return new MascInt(var + a);
                }
        }catch(Exception e) {
```

```
                System.err.println(e);
        }
        float b = var - MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
        return mfloat;
        //return new MascFloat(var + MascFloat.floatValue(dt));
}

public DataType sub(DataType dt) {
        var -= intValue(dt);
        return this;
}

public DataType multiply(DataType dt) {
        /*if(dt instanceof MascInt) {
                return new MascInt(var * intValue(dt));
        }
        return new MascFloat(var * MascFloat.floatValue(dt));*/

        if(dt instanceof MascInt) {
                int b = var * intValue(dt);
                DataType mint = new MascInt(b);
                mint.setName(Integer.toString(b));
                return mint;
                //return new MascInt(var + intValue(dt));
        }
        try {
                Integer a = Integer.parseInt(dt.getName());

                if(a instanceof Integer) {

                        int b = var * a;
                        DataType mint = new MascInt(b);
                        mint.setName(Integer.toString(b));
                        return mint;
                        //return new MascInt(var + a);
                }
        }catch(Exception e) {
                System.err.println(e);
        }
        float b = var * MascFloat.floatValue(dt);
        DataType mfloat = new MascFloat(b);

        mfloat.setName(Float.toString(b));
```

```
		return mfloat;
		//return new MascFloat(var + MascFloat.floatValue(dt));
	}

	public DataType mul(DataType dt) {
		var *= intValue(dt);
		return this;
	}

	public DataType divide(DataType dt) {
		/*if(dt instanceof MascInt) {
			return new MascInt(var / intValue(dt));
		}
		return new MascFloat(var / MascFloat.floatValue(dt));*/

		if(dt instanceof MascInt) {
			int b = var / intValue(dt);
			DataType mint = new MascInt(b);
			mint.setName(Integer.toString(b));
			return mint;
			//return new MascInt(var + intValue(dt));
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {

				int b = var / a;
				DataType mint = new MascInt(b);
				mint.setName(Integer.toString(b));
				return mint;
				//return new MascInt(var + a);
			}
		}catch(Exception e) {
			System.err.println(e);
		}
		float b = var / MascFloat.floatValue(dt);
		DataType mfloat = new MascFloat(b);

		mfloat.setName(Float.toString(b));
		return mfloat;
	}

	public DataType div(DataType dt) {
		var /= intValue(dt);
		return this;
```

```
	}

	public DataType modulus(DataType dt) {
		/*if(dt instanceof MascInt) {
			return new MascInt(var % intValue(dt));
		}
		return new MascFloat(var % MascFloat.floatValue(dt));*/

		if(dt instanceof MascInt) {
			int b = var % intValue(dt);
			DataType mint = new MascInt(b);
			mint.setName(Integer.toString(b));
			return mint;
			//return new MascInt(var + intValue(dt));
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {

				int b = var % a;
				DataType mint = new MascInt(b);
				mint.setName(Integer.toString(b));
				return mint;
				//return new MascInt(var + a);
			}
		}catch(Exception e) {
			System.err.println(e);
		}
		float b = var % MascFloat.floatValue(dt);
		DataType mfloat = new MascFloat(b);

		mfloat.setName(Float.toString(b));
		return mfloat;
	}

public DataType remainder(DataType dt) {
   var %= intValue(dt);
   return this;
}

	public DataType gt(DataType dt) {
		//if(dt instanceof MascInt) {
			//return new MascInt(var > intValue(dt) ? 1 : 0);
			return new MascBool(var > intValue(dt));
		//}
```

```
		/*try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {
				return new MascBool(var > a);
			}
		}catch(Exception e) {
			System.err.println(e);
		}*/
		//return dt.error("> not int"); //new MascFloat(var >
MascFloat.floatValue(dt) ? 1 : 0);
	}

	public DataType ge(DataType dt) {
		if(dt instanceof MascInt) {
			//return new MascInt(var >= intValue(dt) ? 1 : 0);
			return new MascBool(var >= intValue(dt));
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {
				return new MascBool(var >= a);
			}
		}catch(Exception e) {
			System.err.println(e);
		}
		return dt.error(">= not int"); //new MascFloat(var >=
MascFloat.floatValue(dt) ? 1 : 0);
	}

	public DataType lt(DataType dt) {
		if(dt instanceof MascInt) {
			//return new MascInt(var < intValue(dt) ? 1 : 0);
			return new MascBool(var < intValue(dt));
		}
		try {
			Integer a = Integer.parseInt(dt.getName());

			if(a instanceof Integer) {
				return new MascBool(var < a);
			}
		}catch(Exception e) {
			System.err.println(e);
		}
```

```
                return dt.error("< not int"); //new MascFloat(var <
MascFloat.floatValue(dt) ? 1 : 0);
        }

        public DataType le(DataType dt) {
                if(dt instanceof MascInt) {
                        //return new MascInt(var <= intValue(dt) ? 1 : 0);
                        return new MascBool(var <= intValue(dt));
                }
                try {
                        Integer a = Integer.parseInt(dt.getName());

                        if(a instanceof Integer) {
                                return new MascBool(var <= a);
                        }
                }catch(Exception e) {
                        System.err.println(e);
                }
                return dt.error("<= not int"); //new MascFloat(var <=
MascFloat.floatValue(dt) ? 1 : 0);
        }

        public DataType eq(DataType dt) {
                if(dt instanceof MascInt) {
                        //return new MascInt(var == intValue(dt) ? 1 : 0);
                        return new MascBool(var == intValue(dt));
                }
                try {
                        Integer a = Integer.parseInt(dt.getName());

                        if(a instanceof Integer) {
                                return new MascBool(var == a);
                        }
                }catch(Exception e) {
                        System.err.println(e);
                }
                return dt.error("== not int"); //new MascFloat(var ==
MascFloat.floatValue(dt) ? 1 : 0);
        }

        public DataType ne(DataType dt) {
                if(dt instanceof MascInt) {
                        //return new MascInt(var != intValue(dt) ? 1 : 0);
                        return new MascBool(var != intValue(dt));
                }
                try {
```

```
                        Integer a = Integer.parseInt(dt.getName());

                        if(a instanceof Integer) {
                                return new MascBool(var != a);
                        }
                }catch(Exception e) {
                        System.err.println(e);
                }
                return dt.error("!= not int"); //new MascFloat(var !=
MascFloat.floatValue(dt) ? 1 : 0);
        }

        /**
         * @return the var
         */
        public int getVar() {
                return var;
        }

        /**
         * @param var the var to set
         */
        public void setVar(int var) {
                this.var = var;
        }

        public void print(PrintWriter pw) {
                pw.println(Integer.toString(var));
        }

        public void print() {
                System.out.println(Integer.toString(var));
        }
}
```

MascInterpreter

```
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Vector;

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

/***
 *
 * @author Christos
 * Kristina debugged funcInvoke(), getVariable()
```

```
 */

public class MascInterpreter {
	MascSymbolTable st;

	private final static int masc_break = 0;
	private final static int masc_continue = 1;
	private final static int masc_return = 2;

	private int control = -1;

   private String label;

	public MascInterpreter() {
		st = new MascSymbolTable(null);
		registerLibrary();
	}

	public static DataType getNumber(String str) {
		if(str.contains(".") || str.contains("e") || str.contains("E")) {
			Float b = Float.parseFloat(str);
			DataType mfloat = new MascFloat(b);
			mfloat.setName(Float.toString(b));
			return mfloat;
		}
		Integer b = Integer.parseInt(str);
		DataType mint = new MascInt(b);
		mint.setName(Integer.toString(b));
		return mint;
	}

	public DataType regArray(DataType dt) {
		st.put(dt.getName(), dt);
		return dt;
	}

	public static int getDataType(String str) {
		String s = str.replace('e', '0');
		s = s.replace('E', '0');
		String upS, lowS;
		upS = s.toLowerCase();
		lowS = s.toUpperCase();

		if(str.equals("TRUE") || str.equals("FALSE")) {
			return DataType.boolType;
		}
```

```
                if(!upS.equals(lowS)) {
                        return DataType.stringType;
                }
                else {
                        return DataType.getType(getNumber(str));
                }
        }

        public void funcRegister(String name, String[] args, AST body) {
    st.put(name, new MascFunction(name, args, body, st));
}

        public DataType execLibrary(int id, DataType[] params) {
    return MascLibraryFunction.run(st, id, params);
}

public void registerLibrary() {
    MascLibraryFunction.register(st);
}

public DataType rvalue(DataType dt) {
    if(dt.getName() == null) {
        return dt;
    }
    return new DataType(dt.getName());
}

public DataType getVariable(String name) {
    DataType dt = st.get(name);
    if(dt == null) {
        return new DataType(name);
    }
        // hack!  hack!  hack!
        else if(DataType.getType(dt) == DataType.funcType) {
            dt.setName(name);
        }
    return dt;
}

public DataType assignArray(DataType a, DataType b, DataType c) {
        if(a.getName() != null) {
                DataType rval = c;
                String n = "@" + a.getName() + "1";
                DataType lval = st.get(n);
                int type = -1;
```

```
                ArrayList<Integer> indeces = new ArrayList<Integer>();
                Integer index;

                if(b instanceof MascInt) {
                        indeces.add(MascInt.intValue(b));
                }
                else {
                        try {
                                index =Integer.parseInt(b.getName());

                                if(!(index instanceof Integer)) {
                                        throw new MascException("index must be
integer");
                                }
                                else {
                                        indeces.add(index);
                                }
                        }catch(Exception e) {
                                System.err.println(e);
                        }
                }
                rval = st.get(c.getName());
                DataType[] d = new DataType[1];
                d[0] = b;
                MascMultArray arr = new MascMultArray(d, MascString.mString,
a.getName());

                if(rval == null) {
                        MascString ms = new MascString(c.getName().toString());
                        ms.setName(c.getName());
                        arr.set(indeces, ms);

                                DataType dt = arr;
                                dt.setName(n);
            st.put(dt.getName(), arr);
            return dt;
                }
                else if(rval != null) {
                        DataType dd = st.get(c.getName());
                        if(rval instanceof MascInt) {
                                type = DataType.intType;
                        }
                        else if(rval instanceof MascFloat) {
                                type = DataType.floatType;
                        }
                        else if(rval instanceof MascString) {
```

```
                        type = DataType.stringType;
                }
                else if(rval instanceof MascBool) {
                        type = DataType.boolType;
                }

                MascString ms;

                switch(type) {
                case DataType.intType:
                        ms = new
MascString(Integer.toString(((MascInt)st.get(c.getName())).getVar()));
                        arr.set(indeces, ms);
                if(lval != null) {
                        DataType dt = arr;
                        dt.setName(n);
        st.put(dt.getName(), arr);
        return dt;
                }
                break;
                case DataType.floatType:
                        ms = new
MascString(Float.toString(((MascFloat)st.get(c.getName())).getVar()));
                        arr.set(indeces, ms);
                if(lval != null) {
                        DataType dt = arr;
                        dt.setName(n);
        st.put(dt.getName(), arr);
        return dt;
                }
                break;
                case DataType.stringType:
                        ms = new
MascString(((MascString)st.get(c.getName())).getVar());
                        arr.set(indeces, ms);
                if(lval != null) {
                        DataType dt = arr;
                        dt.setName(n);
        st.put(dt.getName(), arr);
        return dt;
                }
                break;
                case DataType.boolType:
                        ms = new
MascString(Boolean.toString(((MascBool)st.get(c.getName())).getVar()));
                        arr.set(indeces, ms);
```

```
                        if(lval != null) {
                                DataType dt = arr;
                                dt.setName(n);
                        st.put(dt.getName(), arr);
                        return dt;
                                }
                                break;
                                default:
                                        return a.error(c, " = ");
                                }
                        }
                }

          return a.error(c, "=");
        }

        public DataType assignArray(DataType a, DataType[] b, DataType c) {
                if(a.getName() != null) {
                        DataType rval = c;
                        DataType lval = st.get(a.getName());
                        int type = -1;
                        ArrayList<Integer> indeces = new ArrayList<Integer>();
                        Integer index;

                        for(int i = 0; i < b.length; ++i) {
                                if(b[i] instanceof MascInt) {
                                        indeces.add(MascInt.intValue(b[i]));
                                }
                                else {
                                        try {
                                                index =Integer.parseInt(b[i].getName());

                                                if(!(index instanceof Integer)) {
                                                        throw new MascException("index must be
integer");
                                                }
                                                else {
                                                        indeces.add(index);
                                                }
                                        }catch(Exception e) {
                                                System.err.println(e);
                                        }
                                }
                        }
                        rval = st.get(c.getName());
                        MascMultArray arr = (MascMultArray)lval;
```

```
                if(rval == null) {
                        MascString ms = new MascString(c.getName());
                        arr.set(indeces, ms);
                        if(lval != null) {
                                DataType dt = arr;
                                dt.setName(a.getName());
                st.put(dt.getName(), arr);
                return dt;
                        }
                }
                else if(rval != null) {
                        MascString ms = new
MascString(((MascString)st.get(c.getName())).getVar().toString());
                        arr.set(indeces, ms);
                        if(lval != null) {
                                DataType dt = arr;
                                dt.setName(a.getName());
                st.put(a.getName(), arr);
                return dt;
                        }
                }
        }

    return a.error( c, "=" );
  }

  public DataType assign(DataType a, DataType b) {
        if(a.getName() != null) {
                DataType rval = b;
                DataType lval = st.get(a.getName());
                int type = -1;

                String start = b.getName();
                boolean sm = start.matches("^\\d");

                boolean intm = false;
                boolean floatm = false;

                try {
                Integer m1 = Integer.parseInt(b.getName());
                if(b instanceof MascInt || m1 instanceof Integer) {
                        intm = true;
                }
                } catch(Exception e) {
```

```
                }

                try {
                Float m2 = Float.parseFloat(b.getName());
                if(b instanceof MascFloat || m2 instanceof Float) {
                        floatm = true;
                }
                } catch(Exception e) {

                }

                if(!((b instanceof MascString || intm || floatm ||
b.getName().equals("TRUE") || b.getName().equals("FALSE")))) {
                        rval = st.get(b.getName());
                        if(rval == null) {
                                throw new MascException(b.getName() + " variable cannot
be found");
                        }

                        DataType dt = rvalue(rval);

                        if(rval instanceof MascInt) {
                                type = DataType.intType;
                        }
                        else if(rval instanceof MascFloat) {
                                type = DataType.floatType;
                        }
                        else if(rval instanceof MascString) {
                                type = DataType.stringType;
                        }
                        else if(rval instanceof MascBool) {
                                type = DataType.boolType;
                        }

                        switch(type) {
                        case DataType.intType:
                                MascInt mint = new
MascInt(((MascInt)st.get(b.getName())).getVar());
                        mint.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mint);
                return dt;
                        case DataType.floatType:
                                MascFloat mfloat = new
MascFloat(((MascFloat)st.get(b.getName())).getVar());
                        mfloat.setName(a.getName());
```

```
                dt.setName(a.getName());
                st.put(dt.getName(), mfloat);
                return dt;
                        case DataType.stringType:
                                MascString mstring = new
MascString(((MascString)st.get(b.getName())).getVar());
                        mstring.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mstring);
                return dt;
                        case DataType.boolType:
                                MascBool mbool = new
MascBool(((MascBool)st.get(b.getName())).getVar());
                        mbool.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mbool);
                return dt;
                        default:
                                return a.error( b, "=" );
                        }
                   }
                   else {
                        type = getDataType(rval.getName());
                        DataType dt = rvalue(rval);

                        switch(type) {
                        case DataType.intType:
                                MascInt mint = new
MascInt(Integer.parseInt(rval.getName()));
                        mint.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mint);
                return dt;
                        case DataType.floatType:
                                MascFloat mfloat = new
MascFloat(Float.parseFloat(rval.getName()));
                        mfloat.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mfloat);
                return dt;
                        case DataType.stringType:
                                MascString mstring = new MascString(rval.getName());
                        mstring.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mstring);
                return dt;
```

```
                        case DataType.boolType:
                                MascBool mbool = new
MascBool(Boolean.parseBoolean(rval.getName().toLowerCase()));
                        mbool.setName(a.getName());
                dt.setName(a.getName());
                st.put(dt.getName(), mbool);
                return dt;
            default:
                return a.error( b, "=" );
                        }
                }
    }

    return a.error( b, "=" );
  }


  public void setBreak(String label) {
    this.label = label;
    control = masc_break;
  }

  public void setContinue( String label ) {
    this.label = label;
    control = masc_continue;
  }

  public void setReturn( String label ) {
    this.label = label;
    control = masc_return;
  }

  public DataType rightDirection(DataType arg1, DataType arg2) {
        String str = ((MascString)arg1).getVar() + " > " + ((MascString)arg2).getVar();
        String out = shellDriver.getshellDriver(str);

        return new MascString(out);
  }

  public DataType leftDirection(DataType arg1, DataType arg2) {
        String str = ((MascString)arg1).getVar() + " < " + ((MascString)arg2).getVar();
        String out = shellDriver.getshellDriver(str);

        return new MascString(out);
  }
```

```
    public DataType rightAppendDirection(DataType arg1, DataType arg2) {
        String str = ((MascString)arg1).getVar() + " >> " + ((MascString)arg2).getVar();
        String out = shellDriver.getshellDriver(str);

        return new MascString(out);
    }

    public DataType pipeline(DataType arg1, DataType arg2) {
        String str = ((MascString)arg1).getVar() + " | " + ((MascString)arg2).getVar();
        String out = shellDriver.getshellDriver(str);

        return new MascString(out);
    }

    public String[] convVarList(Vector v) {
        String[] strArray = new String[v.size()];
        for(int i = 0; i < v.size(); ++i) {
                strArray[i] = (String) v.elementAt(i);
        }

        return strArray;
    }

    public DataType[] convExprList(Vector v) {
        DataType[] dataArray = new DataType[v.size()];
        for(int i = 0; i < v.size(); ++i) {
                dataArray[i] = (DataType) v.elementAt(i);
        }

        return dataArray;
    }

    public DataType funcInvoke(MascAntlrWalker walker, DataType func, DataType[]
params) {

        DataType tp = (DataType) st.get(func.getName());

        if(!(tp instanceof MascFunction)) {
                return func.error("not a function");
        }

        if(((MascFunction)tp).isLibrary()) {
                DataType dt = execLibrary(((MascFunction)tp).getId(), params);
        return dt;
      }
```

```
        String[] args = ((MascFunction)tp).getArgs();
    if(args.length != params.length) {
        return func.error( "unmatched length of parameters" );
    }

    //st.enterScope(((MascFunction)tp).getSymbolTable());

    for (int i = 0; i < args.length; i++) {
        DataType d = rvalue(params[i]);
        d.setName(args[i]);
        System.out.println(args[i]);
        st.put(args[i], d);
    }

    DataType r = null;
                try {
                        r = walker.expr(((MascFunction)tp).getBody());
                } catch (RecognitionException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

    //st.enterScope(((MascFunction)tp).getSymbolTable());

    if (control == masc_break || control == masc_continue) {
          throw new MascException("nowhere to break or continue");
    }

    if(control == masc_return) {
        if(label.equals(((MascFunction)func).getName()) || label == null) {
                control = -1;
        }
    }

    //st.leaveScope();

    return r;
}

public void loopNext(String lp_label) {
        if(control == masc_continue) {
        if(label.equals(lp_label) || label == null) {
                control = -1;
        }
    }
}
```

```
    public void loopEnd(String lp_label) {
        if(control == masc_break) {
            if(label.equals(lp_label) || label == null) {
                    control = -1;
            }
        }
    }

    public boolean canProceed() {
        return control == -1;
    }

    public boolean forCanProceed(DataType expr) {
        if (control != -1) {
            return false;
        }
        if(expr instanceof MascBool) {
            return ((MascBool)expr).getVar();
        }
                    return false;

    }

}
```

MascLibraryFunction

```
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import org.htmlparser.parserapplications.StringExtractor;
import org.htmlparser.util.ParserException;

/***
 *
```

```
 * @author Christos Angelopoulos
 *
 * Kristina - added browser-related functions and implementations 27-30 and 34
 */

public class MascLibraryFunction {
	static Random random = new Random();
	static ArrayList<String> FoList = new ArrayList<String>();
	static HashMap<String, BufferedInputStream> FrHash = new HashMap<String,
BufferedInputStream>();
	static HashMap<String, BufferedOutputStream> FwHash = new
HashMap<String, BufferedOutputStream>();

   final static int f_print = 0;
   final static int f_read = 1;
   final static int f_fopen = 2;
   final static int f_fclose = 3;
   final static int f_random = 4;
   final static int f_min = 5;
   final static int f_max = 6;
   final static int f_round = 7;
   final static int f_floor = 8;
   final static int f_ceil = 9;
   final static int f_abs = 10;
   final static int f_fabs = 11;
   final static int f_sqrt = 12;
   final static int f_exp = 13;
   final static int f_log = 14;
   final static int f_pow = 15;
   final static int f_sin = 16;
   final static int f_cos = 17;
   final static int f_tan = 18;
   final static int f_asin = 19;
   final static int f_acos = 20;
   final static int f_atan  = 21;
   final static int f_fprint = 22;
   final static int f_fread = 23;
   final static int f_match = 24;
   final static int f_getPageText = 25;
   final static int f_getParsedText = 26;
   final static int go = 27; //opens a url in the browser window
   final static int reload = 28; //reloads the current page
   final static int back = 29;
   final static int forward = 30;
   final static int getInt = 31;
   final static int getFloat = 32;
```

```
final static int shell = 33;
final static int extractLink = 34;

public static void register(MascSymbolTable st) {
   st.put("print", new MascFunction( null, f_print));
   st.put("read", new MascFunction( null, f_read));
   st.put("open", new MascFunction( null, f_fopen));
   st.put("save", new MascFunction( null, f_fclose));
   st.put("random", new MascFunction( null, f_random));
   st.put("min", new MascFunction( null, f_min));
   st.put("max", new MascFunction( null, f_max));
   st.put("round", new MascFunction( null, f_round));
   st.put("floor", new MascFunction( null, f_floor));
   st.put("ceil", new MascFunction( null, f_ceil));
   st.put("abs", new MascFunction( null, f_abs));
   st.put("fabs", new MascFunction( null, f_fabs));
   st.put("sqrt", new MascFunction( null, f_sqrt));
   st.put("expl", new MascFunction( null, f_exp));
   st.put("log", new MascFunction( null, f_log));
   st.put("pow", new MascFunction( null, f_pow));
   st.put("sin", new MascFunction( null, f_sin));
   st.put("cos", new MascFunction( null, f_cos));
   st.put("tan", new MascFunction( null, f_tan));
   st.put("asin", new MascFunction( null, f_asin));
   st.put("acos", new MascFunction( null, f_acos));
   st.put("atan", new MascFunction( null, f_atan));
   st.put("fprint", new MascFunction( null, f_fprint));
   st.put("fread", new MascFunction( null, f_read));
   st.put("match", new MascFunction( null, f_match));
   st.put("getPageText", new MascFunction(null, f_getPageText));
   st.put("getParsedText", new MascFunction(null, f_getParsedText));
   st.put("E", new MascFloat((float) Math.E));
   st.put("PI", new MascFloat((float) Math.PI));
   st.put("go", new MascFunction(null, go));
   st.put("reload", new MascFunction(null, reload));
   st.put("back", new MascFunction(null, back));
   st.put("forward", new MascFunction(null, forward));
   st.put("getInt", new MascFunction(null, getInt));
   st.put("getFloat", new MascFunction(null, getFloat));
   st.put("shell", new MascFunction(null, shell));
}

public static DataType run(MascSymbolTable st, int id, DataType[] params) {

      switch(id) {
      case f_print:
```

```
                for(int i = 0; i < params.length; ++i) {
                        params[i].print();
                }
                return null;

        case f_read:
                BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
                String input;
                MascString inp = null;
                        try {
                                input = in.readLine();
                                inp = new MascString(input);
                        } catch (IOException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                return inp;

        case f_fread:
                if(params.length == 1) {
                        String f_name = ((MascString)params[0]).getVar();
                        if(FrHash.get(f_name) == null && FoList.contains(f_name)) {
                                FileInputStream fis = null;
                                String line = null;
                                        try {
                                                fis = new FileInputStream(f_name);
                                                BufferedInputStream bis = new
BufferedInputStream(fis);
                                        DataInputStream dis = new DataInputStream(bis);
                                        try {
                                                        line = dis.readLine();
                                                } catch (IOException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }
                                        FrHash.put(f_name, bis);

                                        } catch (FileNotFoundException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }

                                return new MascString(line);
                        }
```

```
                else if(FoList.contains(f_name)) {
                        String line = null;
                        try {
                                BufferedInputStream bis = FrHash.get(f_name);
                        DataInputStream dis = new DataInputStream(bis);

                                        line = dis.readLine();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                                return new MascString(line);
                        }
                }
                else {
                        throw new MascException("Invalid number of arguments, only
one argument needed");
                }

                case f_fopen:
                if(params.length == 1) {
                        String f_name = ((MascString)params[0]).getVar();
                        if(!FoList.contains(f_name)) {

                                FoList.add(f_name);
                                return new MascString(f_name);
                        }
                        else {
                                throw new MascException("File " + f_name + " is already
opened");
                        }
                }
                else {
                        throw new MascException("Invalid number of arguments, only
one argument needed");
                }

                case f_fclose:
                        if(params.length == 1) {
                        String f_name = ((MascString)params[0]).getVar();
                        if(FoList.contains(f_name)) {
                                Object ob = FrHash.get(f_name);
                                if(ob != null) {
                                        BufferedInputStream bis =
(BufferedInputStream)ob;
```

```
                                        DataInputStream dis = new DataInputStream(bis);
                                        try {
                                                dis.close();
                                        } catch (IOException e) {
                                                // TODO Auto-generated
catch block
                                                e.printStackTrace();
                                        }
                                        FrHash.remove(f_name);
                                }
                                FoList.remove(f_name);

                                return null;
                        }
                        else {
                                throw new MascException("File " + f_name + " cannot be
closed becasue it is not opened");
                        }
                        }
                        else {
                        throw new MascException("Invalid number of arguments, only
one argument needed");
                }

                case f_random:
                        if(params.length == 0) {
                                return new MascFloat(random.nextFloat());
                        }
                        else if(params.length == 1) {
                                if(params[0] instanceof MascFloat) {
                                        random.setSeed((long)
((MascFloat)params[0]).getVar());
                                        return new MascFloat(random.nextFloat());
                                }
                                if(params[0] instanceof MascInt) {
                                        random.nextInt(((MascInt)params[0]).getVar());
                                }
                        }
                        throw new MascException("random accepts 0-1 parameters");

                case f_min:
                        if(params.length == 2) {
                                if(params[0] instanceof MascInt && params[1] instanceof
MascInt) {
                                        int a = ((MascInt)params[0]).getVar();
                                        int b = ((MascInt)params[1]).getVar();
```

```
                                return new MascInt(a > b ? b : a);
                        }

                        else if(params[0] instanceof MascInt && params[1]
instanceof MascFloat) {
                                int a = ((MascInt)params[0]).getVar();
                                float b = ((MascFloat)params[1]).getVar();

                                return new MascFloat(a > b ? b : a);
                        }

                        else if(params[0] instanceof MascFloat && params[1]
instanceof MascInt) {
                                float a = ((MascFloat)params[0]).getVar();
                                int b = ((MascInt)params[1]).getVar();

                                return new MascFloat(a > b ? b : a);
                        }
                        else if(params[0] instanceof MascFloat && params[1]
instanceof MascFloat) {
                                float a = ((MascFloat)params[0]).getVar();
                                float b = ((MascFloat)params[1]).getVar();

                                return new MascFloat(a > b ? b : a);
                        }
                        throw new MascException("Incompatible types for
comparison");
                }
                throw new MascException("Wrong number of arguments, 2
needed");

        case f_max:
                if(params.length == 2) {
                        if(params[0] instanceof MascInt && params[1] instanceof
MascInt) {
                                int a = ((MascInt)params[0]).getVar();
                                int b = ((MascInt)params[1]).getVar();

                                return new MascInt(a < b ? b : a);
                        }

                        else if(params[0] instanceof MascInt && params[1]
instanceof MascFloat) {
                                int a = ((MascInt)params[0]).getVar();
                                float b = ((MascFloat)params[1]).getVar();
```

```
                                return new MascFloat(a < b ? b : a);
                        }

                        else if(params[0] instanceof MascFloat && params[1]
instanceof MascInt) {
                                float a = ((MascFloat)params[0]).getVar();
                                int b = ((MascInt)params[1]).getVar();

                                return new MascFloat(a < b ? b : a);
                        }
                        else if(params[0] instanceof MascFloat && params[1]
instanceof MascFloat) {
                                float a = ((MascFloat)params[0]).getVar();
                                float b = ((MascFloat)params[1]).getVar();

                                return new MascFloat(a < b ? b : a);
                        }
                        throw new MascException("Incompatible types for
comparison");
                }
                throw new MascException("Wrong number of arguments, 2
needed");

        case f_round:
                if(params.length == 1) {
                        float b = Math.round(MascFloat.floatValue(params[0]));
                        DataType mfloat = new MascFloat(b);

                        mfloat.setName(Float.toString(b));
                        return mfloat;
                }
                throw new MascException("round() accepts 1 parameter");

        case f_ceil:
                if(params.length == 1) {
                        float b =
(float)Math.ceil(MascFloat.floatValue(params[0]));
                        DataType mfloat = new MascFloat(b);

                        mfloat.setName(Float.toString(b));
                        return mfloat;
                }
                throw new MascException("ceil() accepts 1 parameter");

        case f_floor:
```

```
                        if(params.length == 1) {
                                float b =
(float)Math.floor(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
        }
                        throw new MascException("floor() accepts 1 parameter");

                case f_abs:
                        if(params.length == 1) {
                                float b = Math.abs(MascInt.intValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new
MascInt(Math.abs(MascInt.intValue(params[0])));
                        }
                        throw new MascException("abs() accepts 1 parameter");

                case f_fabs:
                        if(params.length == 1) {
                                float b = Math.abs(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new
MascFloat(Math.abs(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("fabs() accepts 1 parameter");

                case f_sqrt:
                        if(params.length == 1) {
                                float b = (float)
Math.sqrt(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.sqrt(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("sqrt() accepts 1 parameter");
```

```
                case f_exp:
                        if(params.length == 1) {
                                float b = (float)
Math.exp(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.exp(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("exp() accepts 1 parameter");

                case f_log:
                        if(params.length == 1) {
                                float b = (float)
Math.log(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.log(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("log() accepts 1 parameter");

                case f_pow:
                        if(params.length == 2) {
                                float power = MascFloat.floatValue(params[1]);
                                float b = (float)
Math.pow(MascFloat.floatValue(params[0]), power);
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.pow(MascFloat.floatValue(params[0]), power));
                        }
                        throw new MascException("pow() accepts 2 parameters");

                case f_sin:
                        if(params.length == 1) {
                                float b = (float)
Math.sin(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);
```

```
                        mfloat.setName(Float.toString(b));
                        return mfloat;
                        //return new MascFloat((float)
Math.sin(MascFloat.floatValue(params[0])));
                }
                throw new MascException("sin() accepts 1 parameter");

        case f_cos:
                if(params.length == 1) {
                        float b = (float)
Math.cos(MascFloat.floatValue(params[0]));
                        DataType mfloat = new MascFloat(b);

                        mfloat.setName(Float.toString(b));
                        return mfloat;
                        //System.out.println("cos: " + (float)
Math.cos(MascFloat.floatValue(params[0])));
                        //return new MascFloat((float)
Math.cos(MascFloat.floatValue(params[0])));
                }
                throw new MascException("cos() accepts 1 parameter");

        case f_tan:
                if(params.length == 1) {
                        float b = (float)
Math.tan(MascFloat.floatValue(params[0]));
                        DataType mfloat = new MascFloat(b);

                        mfloat.setName(Float.toString(b));
                        return mfloat;
                        //return new MascFloat((float)
Math.tan(MascFloat.floatValue(params[0])));
                }
                throw new MascException("tan() accepts 1 parameter");

        case f_asin:
                if(params.length == 1) {
                        float b = (float)
Math.asin(MascFloat.floatValue(params[0]));
                        DataType mfloat = new MascFloat(b);

                        mfloat.setName(Float.toString(b));
                        return mfloat;
                        //return new MascFloat((float)
Math.asin(MascFloat.floatValue(params[0])));
```

```
                        }
                        throw new MascException("asin() accepts 1 parameter");

                case f_acos:
                        if(params.length == 1) {
                                float b = (float)
Math.acos(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.acos(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("acos() accepts 1 parameter");

                case f_atan:
                        if(params.length == 1) {
                                float b = (float)
Math.atan(MascFloat.floatValue(params[0]));
                                DataType mfloat = new MascFloat(b);

                                mfloat.setName(Float.toString(b));
                                return mfloat;
                                //return new MascFloat((float)
Math.atan(MascFloat.floatValue(params[0])));
                        }
                        throw new MascException("atan() accepts 1 parameter");

                case f_fprint:
                        if(params.length == 2) {
                        String f_name = ((MascString)params[0]).getVar();
                        String text = ((MascString)params[1]).getVar();
                        if(FwHash.get(f_name) == null && FoList.contains(f_name)) {
                                FileOutputStream fos;
                                        try {
                                                fos = new
FileOutputStream(f_name);
                                                BufferedOutputStream bos = new
BufferedOutputStream(fos);
                                        DataOutputStream dos = new
DataOutputStream(bos);
                                        try {
                                                        dos.writeChars(text);
                                                } catch (IOException e) {
```

```
                                                        // TODO Auto-generated
catch block
                                                        e.printStackTrace();
                                                }
                                        FwHash.put(f_name, bos);
                                                } catch (FileNotFoundException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }


                                        return new MascInt(1);
                                }
                                else if(FoList.contains(f_name)) {
                                        BufferedOutputStream bos = FwHash.get(f_name);
                                        DataOutputStream dos = new DataOutputStream(bos);
                                        try {
                                                        dos.writeChars(text);
                                                } catch (IOException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }

                                        return new MascInt(1);
                                }
                        }
                        else {
                                throw new MascException("Invalid number of arguments, only
one argument needed");
                        }

                        case f_match:
                                if(params.length == 2) {
                                        String str = ((MascString)params[0]).getVar();
                                String regexp = ((MascString)params[1]).getVar();

                                try {

                                        System.out.println(str + " " + regexp);

                                        Pattern pattern = Pattern.compile(regexp);

                                        Matcher matcher = pattern.matcher(str);

                                        //ArrayList<String> strArray = new ArrayList<String>();
```

```
                                String b = matcher.group();
                                DataType mstring = new MascString(b);

                                mstring.setName(b);
                                return mstring;

                                /*while (matcher.find()) {
                                        strArray.add(matcher.group());
                                }

                                MascArray arrMasc = new MascArray(strArray.size(),
DataType.mString);

                                for(int i = 0; i < arrMasc.getSize(); ++i) {
                                        arrMasc.set(i, new MascString(strArray.get(i)));
                                }*

                                return arrMasc;*/
                        } catch(PatternSyntaxException e) {
                                System.err.println("Pattern matching error: " + e);
                        }
                        }
                        else {
                        throw new MascException("Invalid number of arguments, only
two arguments needed");
                }

                case f_getPageText:

                case f_getParsedText:
                        StringExtractor s = new
StringExtractor(((MascString)params[0]).getVar());

                        try {
                                String b = s.extractStrings(false);
                                DataType mstring = new MascString(b);

                                mstring.setName(b);
                                return mstring;
                        } catch (ParserException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
        case go:
           Browser.go(params[0]);
           return new DataType("");
```

```
        case reload:
           Browser.reload();
           return new DataType("");

        case back:
           return new MascBool(Browser.back());
        case forward:
           return new MascBool(Browser.forward());
        case getInt:
                if(params.length == 1) {
                        String b = Integer.toString((MascInt.intValue(params[0])));
                        DataType mstring = new MascString(b);

                        mstring.setName(b);
                        return mstring;
                        //return new
MascString(Integer.toString((MascInt.intValue(params[0]))));
                }
                throw new MascException("getInt() accepts 1 parameter");
        case getFloat:
                if(params.length == 1) {
                        String b = Float.toString((MascFloat.floatValue(params[0])));
                        DataType mstring = new MascString(b);

                        mstring.setName(b);
                        return mstring;
                        //return new
MascString(Float.toString((MascFloat.floatValue(params[0]))));
                }
                throw new MascException("getFloat() accepts 1 parameter");
        case shell:
                if(params.length == 1) {
                        if(params[0] instanceof MascString) {
                                String b = ((MascString)params[0]).getVar();
                                DataType mstring = new MascString(b);

                                mstring.setName(shellDriver.getshellDriver(b));
                                return mstring;
                        }
                        try {
                                String a = params[0].getName();

                                if(a instanceof String) {

                                        DataType mstring = new MascString(a);
```

```
                                        mstring.setName(shellDriver.getshellDriver(a));
                                        return mstring;
                                }
                        }catch(Exception e) {
                                System.err.println(e);
                        }
                }
                throw new MascException("shell accepts 1 parameter");
        case extractLink:
           Browser.extractLink(params[0].getName());
           return new DataType("");
        default:
           throw new MascException("unkown library function");
        }
   }
}
```

MascMultArray

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascMultArray extends DataType {
        private DataType t;
        private int dimensions = 1;
        private ArrayList<Integer> sizeList = new ArrayList<Integer>();
        private DataType[] mArray;
        private int totalSize = 1;

        public MascMultArray(DataType[] dsz, DataType tp, String name) {
                super("@" + name + dsz.length);
                ArrayList<Integer> sz = new ArrayList<Integer>();
                for(int i = 0; i < dsz.length; ++i) {
                        if(dsz[i].getName().matches("\\d+")) {
                                sz.add(Integer.parseInt(dsz[i].getName()));
                        }
                        else {
                                throw new MascException("Sizes must be integers");
```

```
                }
            }
            sizeList = sz;
            t = tp;
            dimensions = sz.size();
            int i = 0;
            int s;
            Iterator<Integer> itInt = sz.iterator();
            while(itInt.hasNext()) {
                ++i;
                s = itInt.next();
                if(s < 0) {
                    throw new MascException("Array size " + i + " " +
sizeList.get(i) + " must be non negative");
                }
                totalSize *= s;
            }
            mArray = new DataType[totalSize];
        }

        public MascMultArray(String sz, DataType tp) {
            //super("[][]");
            StringTokenizer st = new StringTokenizer(sz, ",");
            while(st.hasMoreTokens()) {
                String str = st.nextToken();
                Pattern pattern = Pattern.compile("\\s*");

                Matcher matcher = pattern.matcher(str);
                String sm = matcher.replaceAll("");
                sizeList.add(Integer.valueOf(sm));
            }
            t = tp;
            dimensions = sizeList.size();
            int i = 0;
            int s;
            Iterator<Integer> itInt = sizeList.iterator();
            while(itInt.hasNext()) {
                ++i;
                s = itInt.next();
                if(s < 0) {
                    throw new MascException("Array size " + i + " " +
sizeList.get(i) + " must be non negative");
                }
                totalSize *= s;
            }
            mArray = new DataType[totalSize];
```

```
}

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
	// TODO Auto-generated method stub
	String s = new String("[");
	Iterator<Integer> itInt = sizeList.iterator();
	while(itInt.hasNext()) {
		s += itInt.next() + ", ";
	}
	s += " ] ";
	return s + t.toString();
}

public boolean checkIndex(ArrayList<Integer> indeces) {
	int i = 0;
	int index;
	Iterator<Integer> itInt = indeces.iterator();
	while(itInt.hasNext()) {
		index = itInt.next();
		if((index < 0) && (index >= sizeList.get(i))) {
			return false;
		}
	}

	return true;
}

public int getIndexoutofBounds(ArrayList<Integer> indeces) {
	int i = 0;
	int index;
	Iterator<Integer> itInt = indeces.iterator();
	while(itInt.hasNext()) {
		index = itInt.next();
		if((index < 0) && (index >= sizeList.get(i))) {
			return index;
		}
	}

	return -1;
}

public int getSizeoutofBounds(ArrayList<Integer> indeces) {
```

```
		int i = 0;
		int index;
		Iterator<Integer> itInt = indeces.iterator();
		while(itInt.hasNext()) {
			index = itInt.next();
			if((index < 0) && (index >= sizeList.get(i))) {
				return sizeList.get(i);
			}
		}

		return -1;
	}

	public DataType get(ArrayList<Integer> indeces) {
		int index = 1;
		if(checkIndex(indeces)) {
			index = indeces.get(indeces.size() - 1) * sizeList.get(sizeList.size()
- 2) + indeces.get(indeces.size() - 2);
			for(int i = sizeList.size() - 3; i >= 0; --i) {
				index *= sizeList.get(i);
				index += indeces.get(i);
			}

			return mArray[index];
		}
		index = getIndexoutofBounds(indeces);
		throw new MascException("index out of bounds: " + index + " size: " +
getSizeoutofBounds(indeces));
	}

	public void set(ArrayList<Integer> indeces, DataType dt) {
		int index;
		if(!checkIndex(indeces)) {
			index = getIndexoutofBounds(indeces);
			throw new MascException("index out of bounds: " + index + "
size: " + getSizeoutofBounds(indeces));
		}
		if(indeces.size() > 1) {
			index = indeces.get(indeces.size() - 1) * sizeList.get(sizeList.size()
- 2) + indeces.get(indeces.size() - 2);
		}
		else {
			index = indeces.get(0);
		}
		for(int i = sizeList.size() - 3; i >= 0; --i) {
			index *= sizeList.get(i);
```

```
                    index += indeces.get(i);
                }
                if(t instanceof MascBool) {
                    if(dt instanceof MascBool) {
                        mArray[index] = dt;
                    }
                }
                if(t instanceof MascInt) {
                    if(dt instanceof MascInt) {
                        mArray[index] = dt;
                    }
                    MascInt mint = new MascInt(MascInt.intValue(dt));
                    mArray[index] = (DataType)mint; //MascInt.intValue(dt);
                }
                if(t instanceof MascFloat) {
                    if(dt instanceof MascFloat) {
                        mArray[index] = dt;
                    }
                    MascFloat mfloat = new MascFloat(MascFloat.floatValue(dt));
                    mArray[index] = (DataType)mfloat; //MascInt.intValue(dt);
                }
                if(t instanceof MascString) {
                    if(dt instanceof MascString) {
                        mArray[index] = dt;
                    }
                    if(dt instanceof MascInt) {
                        Integer mint = MascInt.intValue(dt);
                        MascString mstring = new MascString(mint.toString());

                        mArray[index] = (DataType)mstring;
//MascInt.intValue(dt);
                    }
                    if(dt instanceof MascFloat) {
                        Float mfloat = MascFloat.floatValue(dt);
                        MascString mstring = new MascString(mfloat.toString());

                        mArray[index] = (DataType)mstring;
//MascInt.intValue(dt);
                    }
                }
        }

        /**
         * @return the t
         */
        public DataType getT() {
```

```
		return t;
}

/**
 * @param t the t to set
 */
public void setT(DataType t) {
		this.t = t;
}

/**
 * @return the mArray
 */
public DataType[] getMArray() {
		return mArray;
}

/**
 * @param array the mArray to set
 */
public void setMArray(DataType[] array) {
		mArray = array;
}

/**
 * @return the dimensions
 */
public int getDimensions() {
		return dimensions;
}

/**
 * @param dimentions the dimentions to set
 */
public void setDimensions(int dimensions) {
		this.dimensions = dimensions;
}

/**
 * @return the sizeList
 */
public ArrayList<Integer> getSizeList() {
		return sizeList;
}

/**
```

```
	 * @param sizeList the sizeList to set
	 */
	public void setSizeList(ArrayList<Integer> sizeList) {
		this.sizeList = sizeList;
	}

	/**
	 * @return the totalSize
	 */
	public int getTotalSize() {
		return totalSize;
	}

	/**
	 * @param totalSize the totalSize to set
	 */
	public void setTotalSize(int totalSize) {
		this.totalSize = totalSize;
	}
}
```

MascString

```
/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascString extends DataType {

	private String var;

	public MascString(String s) {
		super(s);
		var = s;
		// TODO Auto-generated constructor stub
	}

	public DataType plus(DataType dt) {
		if(dt instanceof MascString) {
			return new MascString(var + ((MascString)dt).getVar());
		}

		return error("string +");
	}

	/**
```

```
	 * @return the var
	 */
	public String getVar() {
		return var;
	}

	/**
	 * @param var the var to set
	 */
	public void setVar(String var) {
		this.var = var;
	}

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

}
```

MascSymbolTable

```
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Stack;

/***
 *
 * @author Christos Angelopoulos
 *
 */

public class MascSymbolTable extends Hashtable<String, DataType> {
	/**
	 *
	 */
	private static final long serialVersionUID = -7074040758384638347L;
	private MascSymbolTable static_next;
	private Stack<MascSymbolTable> MascStack;


	public MascSymbolTable(MascSymbolTable st) {
		static_next = st;
		 MascStack = new Stack<MascSymbolTable>();
	}

	/**
	 * @return the static_next
	 */
```

```
	public final MascSymbolTable getStatic_next() {
		return static_next;
	}

	/**
	 * @param static_next the static_next to set
	 */
	public void setStatic_next(MascSymbolTable static_next) {
		this.static_next = static_next;
	}

	public final boolean contains(String name) {
		return containsKey(name);
	}

	public void enterScope(MascSymbolTable mst) {
		MascStack.push(mst);
	}

	public MascSymbolTable leaveScope() {
		return MascStack.pop();
	}

	public DataType getValue(String name) {
		Stack<MascSymbolTable> temp = (Stack<MascSymbolTable>)
this.MascStack.clone();
		MascSymbolTable st;
		while(!temp.empty()) {
			st = temp.pop();
			Object ob = st.get(name);

			if(ob != null) {
				DataType dt = (DataType)ob;
				return dt;
			}
		}

		return null;
	}

	public void setValue(String name, DataType data) {
		Stack<MascSymbolTable> temp = (Stack<MascSymbolTable>)
this.MascStack.clone();
		MascSymbolTable st;

		while(!temp.empty()) {
```

```
                        st = temp.pop();
                        Object ob = st.get(name);

                        if(ob != null) {
                                st.put(name, data);
                                return;
                        }
                }

                this.put(name, data);
        }

}
```

ShellDriver

```
/* Kristina - wrote original version of shellDriver, execed a process the user
 * entered and redirected the output (Java doesn't use stdout for a forked
 * process, it has to be captured by process.getInputStream (go figure...)
 * and written.  Endlessly attempted to get process.getOutputStream working
 * (to be able to feed stdin into processes like cat) to no avail.
 */

import java.io.*;

/***
 *
 * @author Christos Angelopoulos
 *
 */

public class shellDriver {
        static InputStreamReader in;
   static StringBuffer buff;

   public static String getshellDriver(String args) {
        int returnCode = -1;
        final String output = new String();
      try {
        String os = System.getProperty("os.name");
        String shell = "sh";
        String c = "-c";
        if(os.contains("Mac") || os.contains("Unix") || os.contains("Linux")) {
                shell = "sh";
                c = "-c";
        }
        else if(os.contains("Win")) {
                shell = "cmd";
```

```
            c = "/c";
    }

    in = new InputStreamReader(System.in);
    String cmd = args;
    /*for(int i = 0; i < args.length; ++i) {
            cmd += (args[i] + " ");
    }*/
    String[] command =  { shell, c, cmd };
    final Process process = Runtime.getRuntime().exec(command);
    new Thread() {
            public void run() {
                    try {
                            InputStream is = process.getInputStream();
            byte[] buffer = new byte[1024];
            for(int count = 0; (count = is.read(buffer)) >= 0;) {
                  System.out.write(buffer, 0, count);
            }
                    }
                    catch(Exception e) {
                            e.printStackTrace();
          }
            }
    }.start();
    new Thread() {
            public void run() {
                    try {
                            InputStream is = process.getErrorStream();
            byte[] buffer = new byte[1024];
            for(int count = 0; (count = is.read(buffer)) >= 0;) {
                    //System.err.write(buffer, 0, count);
                    String str = new String(buffer);
                    output.concat(str);
            }
                    }
                    catch(Exception e) {
                            e.printStackTrace();
                    }
       }
    }.start();

    returnCode = process.waitFor();
    in.close();
}
catch (Exception e) {
    e.printStackTrace();
```

```
        }

        return output;
    }

        public static void printPrompt() {
                System.out.print(" >>> ");
        }

        public static String getCmd() {
                try {
                   buff = new StringBuffer("");
                   char ch;
                   while((ch = (char)in.read()) != '\n') {
                         buff.append(ch);
                   }
                }
                catch (IOException e) {
                   System.out.println("Couldn't read input!");
                }
                return buff.toString();
        }
}
```

Walker

```
//*********************************************************************
***
//* Antlr MASC AST Walker
//*
//* File:          walker.g
//* Author:        Vaibhav Saharan, vs2220@columbia.edu
//* Description:    Traverses the tree generated by MascAntlrParser.java
//*               Creates the AntlrMascWalker.java file
//*
//*********************************************************************
****/

{
import java.util.*;
}

class MascAntlrWalker extends TreeParser;
options{
   importVocab = MascAntlr;
}

{
```

```
        static DataType null_data = new DataType("NULL");
    MascInterpreter ipt = new MascInterpreter();


        public void uponEOF() throws Exception {
       System.out.println("EOF");
    }

    public static void main(String[] args) throws Exception {
       MascAntlrLexer lexer = new MascAntlrLexer(System.in);
       MascAntlrParser parser = new MascAntlrParser(lexer);
       // Parse the input expression
       parser.program();
       AST t = parser.getAST();
       DataType dt = new DataType();
       if(t == null) {
          System.out.println("AST is null!");
       }
       else {
          printTree(t, "");
       }

       MascAntlrWalker walker = new MascAntlrWalker();
       dt = walker.expr(t);
       System.out.println("Walker Output:"+dt);
    }

    public static void printTree(AST t, String tabs) {
       int x = t.getNumberOfChildren();
       System.out.println(tabs+t.toString()+": "+x);
       if(x > 0) {
          printTree(t.getFirstChild(), tabs+"\t");
       }

       if(t.getNextSibling() != null) {
          printTree(t.getNextSibling(), tabs);
       }
    }
}

expr returns [DataType dt]
{
    DataType a, b, c;
    Vector v;
    DataType[] x;
    String s = null;
```

```
    String[] sa;
    dt = null_data;
}
    : #(OR a=expr right_or:.)
      {
        if ( a instanceof MascBool )
          dt = ( ((MascBool)a).getVar() ? a : expr(#right_or) );
        else
          dt = a.or( expr(#right_or) );
      }
    | #(AND a=expr right_and:.)
      {
        if ( a instanceof MascBool )
          dt = ( ((MascBool)a).getVar() ? expr(#right_and) : a );
        else
          dt = a.and( expr(#right_and) );
      }
    | #(NOT a=expr)             { dt = a.not(); }
    | #(GTE a=expr b=expr)      { dt = a.ge( b ); }
    | #(LTE a=expr b=expr)      { dt = a.le( b ); }
    | #(GT a=expr b=expr)       { dt = a.gt( b ); }
    | #(LT a=expr b=expr)       { dt = a.lt( b ); }
    | #(EQ a=expr b=expr)       { dt = a.eq( b ); }
    | #(NE a=expr b=expr)       { dt = a.ne( b ); }
    | #(PLUS a=expr b=expr)     { dt = a.plus( b ); }
    | #(MINUS a=expr b=expr)    { dt = a.minus( b ); }
    | #(STAR a=expr b=expr)     { dt = a.multiply( b ); }
    | #(DIV a=expr b=expr)      { dt = a.divide( b ); }
    | #(ASG a=expr b=expr)      { dt = ipt.assign( a, b ); }
    | #(REDIR_IN a=expr b=expr)  { dt = ipt.leftDirection( a, b ); }
    | #(REDIR_OUT a=expr b=expr) { dt = ipt.rightDirection( a, b ); }
    | #(REDIR_APP a=expr b=expr) { dt = ipt.rightAppendDirection( a, b ); }
    | #(PIPE a=expr b=expr)     { dt = ipt.pipeline( a, b ); }
    | num:NUMBER                { dt = ipt.getNumber( num.getText() ); }
    | str:STRING                { dt = new MascString( str.getText() ); }
    | "true"                    { dt = new MascBool( true ); }
    | "false"                   { dt = new MascBool( false ); }
    | #(ARRAY a=expr b=expr c=expr) { dt = ipt.assignArray(a, b, c); }
    | #("if" a=expr thenp:. (elsep:.)?)
      {
        if ( !( a instanceof MascBool ) )
          return a.error( "if: expression should be bool" );
        if ( ((MascBool)a).getVar() )
          dt = expr( #thenp );
        else if ( null != elsep )
          dt = expr( #elsep );
```

```
        }
      | #(CMD (cmd:. { if ( ipt.canProceed() ) dt = expr(#cmd); } )*)
      | #("while" loopbody:.
            (loopid:ID        { s = loopid.getText(); }
            )?
        )
         {
            while ( ipt.canProceed() )
            {
               dt = expr( #loopbody );
               ipt.loopNext( s );
            }
            ipt.loopEnd( s );
        }
      | #("break" (breakid:ID      { s = breakid.getText(); }
              )?
        )                        { ipt.setBreak( s ); }
      | #("continue" (contid:ID    { s = contid.getText(); }
                )?
        )                        { ipt.setContinue( s ); }
      | #("return" ( a=expr        { dt = ipt.rvalue( a ); }
              )?
        )                        { ipt.setReturn( null ); }
      | #("function" fname:ID sa=vlist fbody:.)
        { ipt.funcRegister( fname.getText(), sa, #fbody ); }
      | #(FUNC_CALL a=expr x=mexpr)
        { dt = ipt.funcInvoke( this, a, x ); }
      | #("array" aname:ID         { v = new Vector(); }
         (a=expr                 { v.add( a ); }
         )*
        )
         { dt = new MascMultArray(ipt.convExprList(v), DataType.mString,
aname.getText());
           dt = ipt.regArray(dt) ;}
      | #(id:ID                   { dt = ipt.getVariable( id.getText() ); }
        )
      ;


vlist returns [ String[] sv ]
{
   Vector v;
   sv = null;
}
      : #(VAR_LIST     { v = new Vector(); }
          (s:ID      { v.add( s.getText() ); }
```

```
            )*
          )            { sv = ipt.convVarList( v ); }
        ;


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