# BROOM

*A Matrix Manipulation Language*

**By:**
*Chris Tobin*
*Michael Weiss*
*Gabriel Glaser*
*Brian Pellegrini*

# Chapter 1:
# BROOM: An Introduction

## *1.1 Introduction*

The BROOM programming language is built for flexible, high-level matrix manipulation. It supports a powerful set of built in operations and functions. A flexible set of tools is included within the language to allow a user to define their own matrix operations. An intuitive design allows the BROOM programming language to be learned over the course of an hour. BROOM also supports a subset of most popular programming language constructs of control flow and user-defined functions.

BROOM is designed to be a translated language, with its target language being Java. The user is able to define a set of operations in the form of a program in a text file. The translator will then output a Java source file that can be edited and compiled into Java byte-code. The BROOM translator is equipped with error detection so that BROOM syntactical and semantical errors are not passed on to the Java source code. This also allows for error-checking and debugging algorithms to be a relatively smooth process. Since BROOM translates directly to Java and uses a Java-implemented translator, BROOM is a highly portable.

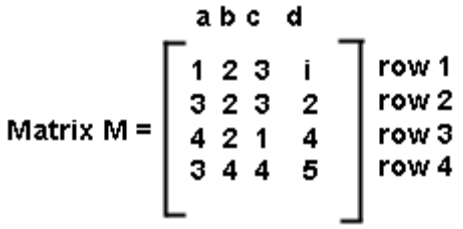
Code definitions in BROOM are structured to be modular. The syntax is friendly and oriented towards matrix operations. BROOM is designed to be consistent with standard methods and naming of matrix operations. If a user familiar with matrix manipulation were to look at a program written in BROOM, the function of that program would be intuitively clear to the user.

---

## *1.2 Background*

A matrix is a mathematical construct that is used in a vast array of areas and professions including: Linear Algebra, Computer Graphics, Internet Search Algorithms, Graph Theory, Forest Management and Electrical Networks. This extremely wide range of applications makes matrices and their operations an incredible powerful tool.

At its most basic and abstract level, a matrix is simply a set of numbers ordered into rows and columns (as shown below):




This collection of numbers can take on any number of meanings for both theoretical and real world problems. For example, the rows can be sets of coefficients of a system of simultaneous equations with each column representing an independent variable. The columns or rows can also represent the vectors that span a certain subspace. On the other hand, the columns and rows could represent the equation needed to solve a least squares regression.

Regardless of the meaning or particular application of a matrix, all matrices share a common set of functions and operations. These functions include matrix multiplication, determinants, Gaussian elimination, inversion, addition, subtraction, scalar multiplication and others. The information that these functions and operations yield can give great insight into the problem that the matrix represents. However, these operations and functions (and their algorithms) are quite complex and time-consuming to implement without the aid of artificial computation. Even the use of a scientific calculator would not ease the tedium and computational complexity of these manipulations. Therefore, given the wide-range applicability and power of matrices and their functions, a language that could easily manipulate and calculate matrices would be a very powerful tool.

---

## *1.3 Related Work*

Most of today's existing matrix manipulation tools are contained as parts of larger and expensive mathematical software. Some examples of these large programs with matrix manipulation capabilities are Mathematica, Matlab, and Octave. These tools have been consulted to observe the popular syntax used in declaring and manipulating matrices. While these tools are powerful, they have high learning curves, difficult function declarations, and excess functionality that is unnecessary for those interested solely matrix manipulation. Even Octave, a freeware GNU project, has a 250 page user's manual.

---

## *1.4 Motivating Scenarios*

Poindexter is working on an assignment for a linear algebra class. His just worked out several examples by hand and wishes to check his answers. He does not have access to an expensive tool such as Matlab and wishes to check his work without having to learn any syntax. He downloads BROOM and within 5 minutes he has checked all of his work, without having to spend hours browsing through a 250 page instruction manual for a professional matrix manipulation language.

A graduate student has data from his research in a large number of matrices that he plans to analyze with his own matrix operation intensive algorithms. Not having prior experience with programming, he does not wish to learn the syntax and rules of a large, extensive program. He uses BROOM to quickly implement these algorithms. BROOM shares enough similarities to Matlab to make a good starting ground for students from any background.

Our language is designed to solve not only these simple, common problems, but more advanced problems as well.

---

## *1.5 Goals*

BROOM is meant to be a syntactically intuitive, high level, portable, flexible language for mathematical manipulations and computations of matrices.

1) **Syntactically Intuitive** - One of the main goals of BROOM is that its syntax should be easy to learn, read, write, and manipulate. BROOM's syntax is meant to be simple and intuitively deducible from a simple knowledge of matrix operations and linear algebra. If a user were to have knowledge of theoretical matrix mathematics and linear algebra, that user should be able to learn to write and understand BROOM code quickly, without extensive programming knowledge.

2) **High Level** - Users do not have to worry about internal representation of data and algorithm implementation. The modularity capabilities allow for expansion to complex problem solving routines.

3) **Portable** - Since BROOM is translated language with its target language as Java and its translator implemented in Java, all BROOM code can be translated, executed, and evaluated on any machine that has a Java Runtime Environment with compiler. Java is a highly portable language, which therefore makes the translator and the BROOM code that it translates highly portable as well.

4) **Flexible** - Through BROOM's modularization and implementation of matrix manipulation functions, the user can create more complex routines with these modules as building blocks. The resulting code is not only very readable, but also flexible.

---

## *1.6 Basic Features*

1) **Control Flow** – BROOM provides users with basic level programming language control of events using conditionals. The statements for flow control include: if-then-else, for loops, and while loops.

2) **Flexible Matrix Definitions** – with BROOM, matrix definition closely resembles the highly proven format in Matlab and matrices can be constructed relatively effortlessly.

3) **Data types** – two types of data are supported. These data types are real numbers (Number) and matrices (Matrix).

4) **Built-In Functions**- BROOM also contains a wide range of built in functions including Gaussian elimination, rank, inversion, transpose, and determinants. However, the user could theoretically implement these operations in the language with the basic constructs.

5) **User-Defined Functions** – BROOM allows the user to define their own functions which makes their resulting code more modular, powerful, and reusable.

## *1.7 Primary Models of Computation*

The primary model of computation for BROOM is a Java implemented translator. ANTLR was used to create the lexer and parser which in turn creates the Abstract Syntax Tree. Both the Static Semantic Analyzer and Code Generator are written as AST Tree Walkers using antler. The basic Matrix types, Number types, basic operators, and built-in functions will be written in a collection of Java source code. User-defined functions are translated from BROOM to Java methods. The final output of the translator will be a Java file of a user-defined title that can be compiled and run to produce the desired behavior of the inputted BROOM program.

# Chapter 2

# BROOM: A Tutorial

If you are familiar with Matrices and simple programming constructs, this tutorial will help you to write useful programs in BROOM in no time. A BROOM program has 4 main parts: program name, global variable declarations and assignments, function definitions, start function definition.

## *2.1 Simple Example*

Remember that BROOM is a translated language, so when you run BCC (the translator) on your source-file, you will get back a Java file which you must compile and run. The first thing that you define in your BROOM program is the name of this file, called the Program Name.

```
MyBroomProgram            //Program Name!

start()                   //start function

      Matrix myMatrix;     //local variable declarations
      Number num1, num2;

        //beginning of statements that
        //should be executed

      myMatrix = [1,2;     //assign your matrix
                  3,4;];
      num1 = det(myMatrix) //let num1 be determinant of matrix
      num2 = 5;            //let num2 be 5
      num1 = num1 + num2;  //num1 now 5 + determinant of matrix
      myMatrix.print();    //print the matrix
      print("The inverse of that matrix + 5 is :");
      num1.print();
endstart
```

This very simple example illustrates the basic layout and functionality of BROOM code. The "start" function is the function that is going to be run when you start up your compiled Java file. So everything that you define inside your start function will be run when you run your program. Notice that the first thing in the start function is the declarations of local variables. If you are going to use variables to store data, you need to declare them at the beginning of the function. Once the variables have been declared, you are free to assign them to valid type-agreeing arguments (e.g. in our example where we assign myMatrix to the 2x2 matrix shown above). The last 4 lines of this program show how output to standard out is written. In this case, we print myMatrix with the

.print() call. Then we print text with the print(". .") call. Finally we print the Number num1 with another .print() call. All of these calls will cause the appropriate text to be printed to the screen.

This example above illustrates a simple BROOM program without function calls and without control flow (i.e. if statements, for loops, etc.). For a more in depth example, see section 2.3. Once you feel comfortable with the above example and have practiced writing a few simple programs, the transition should be easy to section 2.3

## *2.2 Translating & Compiling: Finally Make Something Happen*

Now that you have a source code file written, save it to a file with extension ".broom" (e.g. mycode.broom). The next step is to translate the BROOM file into a java file.

**1) Translate:**

```
$ java BCC <yourprogram.broom>
```

This call will invoke the Java implemented translator BCC that will translate your program into a java file. The outputted Java file will have the name that you specified as the first thing in the .broom source-code. So in our example, running BCC on our source file will generate a java file called: MyBroomProgram.java

Now that you have a java file that is the equivalent of your .broom file, you want to compile it into java byte-code. For this you will need to have a JVM and compiler on your machine (preferable 1.4.1). All that's now left is to compile the .java file:

**2) Compile:**

```
$ javac MyBroomProgram.java
      <youroutputtedprogramname.java>
```

This will now give you a java class file which is an executable. Execute the class file via:

**3) Execute:**

```
$ java MyBroomProgram
      <youroupputtedprogramname>
```

This call will run your desired program.

Note: In order for this process to function as above, you need to make sure that your Java compiler is set up properly and that the BroomMatrix.java library is in the same folder as the source-code.

## *2.3 A More Complex Example: Control Flow and Functions*

The following example is given to show the more complicated features of the BROOM language. If you are a relatively experienced programmer, this should be second nature to you. If you are not, don't worry, BROOM is a simple language and the following constructs might take a little getting used to, but are relatively easy to understand and manipulate:

```
MyNewBroomProgram
//what follows are global variable declarations
global Matrix globalMat;
global Number globalNum;

//global variable assignments
globalMat = [1,2;
             3,4;]
globalNum = 7;

//user defined function with control flow
//function multiplies 2 Matrices and a Number together only if
//the number is not 0
define function Void matrixScalarMult(Matrix a, Matrix b, Number c)
      //local variable declarations and assignments,
      //only seen inside this function
      Number unused1;
      unused1 = 0;

      //check if c is 0, if it is, print that you must get non-zero
      //number as an argument
      if(c == 0)then
            print("You must give a non-zero Number argument");
      else
            a= a*b;
            a = a*c
            print("Below is the result of matrixScalarMult");
            //print the Matrix
            a.print();
      endif
endfunction

//now define the start function which defines what happens when program
//is run.  It simply calls the matrixScalarFunction
start()

      //call matrixScalarMult on the globalMat with a hardcoded matrix
      matrixScalarMult(globalMat, [1,1;2-3,4], globalNum);
endstart
```

The above example shows how global variables (`globalMat` and `globalNum`) can be referenced anywhere in the program (they are used as arguments in `start`()). This program also shows how you assign these variables outside of all function definitions. The next interesting thing that this program introduces is the idea of a *function*. A function is a piece of code that a user defines to take some variable number of arguments and which returns some type (`Matrix`, `Number`, or `Void`). This function can then be called later in the program. Calling a function means running the function code on the arguments provided to it. Finally, this example introduces control-flow in the form of an if statement. The program comments state what the if statement actually checks. For more information on control-flow and functions, please see the Reference Manual (Ch. 2).

# Chapter 3:

# Language Reference Manual

## *3.1 Lexical Conventions*

Each BROOM program consists of a file which can be written in DOS, UNIX, or MAC standard file format using ASCII or UNICODE text. BROOM is case sensitive. There are 4 types of basic tokens that comprise a program. These are identifiers, keywords, operators, and basic separators. Blank spaces, newlines, comments, and tabs are necessary between any adjacent tokens of the same type. These non-token characters are denoted as whitespace. Hence, the general format of the language is free form. The detailed specifications of the lexical conventions of the language are defined below, as well as fully specified in the grammar of the language appended to the end of this document.

### 3.1.1 Whitespace

The DOS, UNIX, and MAC standards are all accepted as whitespace. That is, carriage returns, newlines, and combinations will all be treated as whitespace.

Comments:

BROOM uses one type of commenting style. A comment can span only a single line. A comment begins with '//' and ends with a newline. All text after the '//' and before the newline are ignored by the parser:

*Comment*

```
. . .
//this is a comment, the parser will not look at this at all
Matrix a. . . .
```

### 3.1.2 The Tokens

Outlined below are the lexical constructs for the basic token types in the BROOM programming language. Their significance will be discussed shortly.

Identifiers

Identifiers are formally defined as any sequence of letters (a-z, A-Z) or digits (0-9) that do not begin with a digit and are not a keyword. These are used to represent user defined variables and functions.

*Grammar Rule:*

*ID : ('a'..'z''A'..'Z') ( ('a'..'z'A'..'Z'') | DIG )*;*

Keywords

The BROOM programming language reserves the following words for built in functions:

| | | | |
|---|---|---|---|
| **Number** | **gauss** | **numRows** | **endstart** |
| **Matrix** | **inv** | **numCols** | **print** |
| **if** | **det** | **setRow** | **addRow** |
| **else** | **trans** | **setCol** | **addCol** |
| **then** | **rank** | **function** | **deleteRow** |
| **endif** | **rowSwap** | **endfunction** | **deleteCol** |
| **while** | **colSwap** | **return** | **appendUD** |
| **endwhile** | **getRow** | **global** | **appendLR** |
| **for** | **getCol** | **define** | |
| **endfor** | **getElement** | **start** | |

**NOTE:** BROOM also reserves all **Java reserved words**. Since the BROOM compiler, BCC, translates into Java, no BROOM program can use the Java reserved words as identifiers.

Operators

The following symbols each have unique functions in BROOM programming language:

**Arithmetic Operators :** + - *
**Boolean Operators :** ! && ||
**Logical Operators :** != < > ==

Separators

( ) ; [ ] ,

These are used to build expressions and matrices.

## *3.2 BROOM Programs and Programming Constructs*

(If unsure about the grammar notation, see footnote on ANTLR notation.)[1]

The total grammar for BROOM can be seen in the Appendix under the code listing section. See the file *broom.test.g (by Chris Tobin)* for the entire grammar.

---

1A Note on Grammar Notation

All grammar symbols are defined in a standard ANTLR acceptable notation. These standards are as follows: ( ... ) denotes a sub rule. ( ... ) * denotes a Kleene closure operation on the enclosed expression within the brackets. ( ... ) + denotes a positive closure. ( ... ) ? indicates that the enclosure is optional. A logical OR that abbreviates 2 productions or serves as the union operator in a regular expression is denoted '|'. If we take the range of a set of values, for example, from numbers 2 through 7, the rank is abbreviated with 2 dots, '2' .. '7'. A colon denotes a rule start. And a semi-colon represents the termination of any given rule. For example, the rule A -> B, is then denoted, A : B ; in ANTLR form.

### 3.2.1 Program Layout

A BROOM program consists of a program name, a set of global variable declarations, immediately followed by global variable assignments. These are followed by a list of function definitions. Lastly, the start function is defined. This is the function that will actually define the program flow (what functions will be called, etc). The general layout of a BROOM program follows below:

*{ Identifier }* //the name of the program
*{ Global Variable Declarations ... }* //optional
*{ Global Variable Assignments ... }* //optional
*{ Function definitions ... }* //optional
***{ Start Function (mandatory) }***

**Note**: Each function definition has the same general format as the larger program. See section 2.7 where this is more specifically defined.

### 3.2.2 The 2 Data Types: Matrix and Number

There are only two data types in BROOM: `Number`**'s** and `Matrix`'s:

#### 3.2.2.1 `Number`

`Numbers` are high precision floating point numbers, which have a flexible definition. They are formally defined by the grammar below:

```
signedDouble : (SUB NUMBER) | NUMBER ;
NUMBER :              DECIMAL ( 'e' (SUB)? DECIMAL )? ;
protected DECIMAL :   (DIG)+ (DOT (DIG)+)?;
protected DIG :       ('0'..'9');
DOT:                  '.';
SUB:                  '-';
```

Examples:
1 1.2 -1.25 -0.555e17 1e-2 1.74e1.23

#### 3.2.2.2 `Matrix`

A `Matrix` is a two dimensional array of numbers. A Matrix has **m** rows and **n** columns. Every explicitly (hard-coded) defined `Matrix` has a specific form. Each `Matrix` has at least one `row` whose elements are separated by commas. A semicolon terminates every `row`.

Example `Matrix row`**:**

**1, 2, a-b, det(someMatrix), 5*b ;**

As seen above, the elements of each `row` are not restricted to only `Numbers` or identifiers. Each element can be an `expression` whose evaluation yields a number. `Expression's` are defined later in the manual, but a loose rule is that an

`expression` that can be used as an element of a `row` can be a: `Number`, identifier, function call (that returns a `Number`), arithmetical expression that operates on any of the above, provided that it yields a `Number`. (Note: Static Semantic Analysis will ensure that only the aforementioned `expressions` can be used in a `row` declaration.)

The actual definition of a `Matrix` is one or more `rows`, of equal length, enclosed within square bracket. While the grammar rules are not capable of ensuring equality of row lengths, Static Semantic Analysis ensures that this is enforced.
Example `Matrix` definitions:

```
[1, 2, 3, 4;
 5, 6, 7, 8;
 9, 10, 11, 12;]

[a, b, c;
 det(someMatrix),rank(anotherMatrix), b*c+1;]

[1;]

[1; 2; 3;]
```

The rule that is used to define how a Matrix can be written explicitly in BROOM is shown below (this is not the exact rule from the ANTLR grammar, it has been modified to be more readable):

```
matrix:
        '['! (matrixrow)+ ']'!
        ;
matrixrow: (expr) (','! expr)*  ';'!
        ;
```

### 3.2.3 Variables

A variable is an `Identifier` which refers to a `Matrix` or a `Number`. Variables must always be declared before being assigned or manipulated. There are two types of variables: global and local. A global variable is declared in the beginning of the program. A local variable is be declared at the beginning of a function body. These two types of variable declarations are defined formally as follows:

```
globalvariabledec : "global"! declaration;
declaration : ("Matrix"|"Number") ID (COMMA! ID)* SEMI!
```

These rules allow for more than one variable of the same type to be declared in the same declaration as shown in the following examples:

Example Global Variable Declarations:

```
global Matrix myMatrix, yourMatrix, c, d;
global Number a, b;
```

```
global Number anotherNumber;
```

(Local Variable Declarations are the same as global except they are not preceded by the "global" keyword).

#### 3.2.3.1 Scope of Variables

Given that there are two types of variables, global and local, each must clearly have its own scope. A global variable can be referenced, viewed, and modified from any function in the program. Local variables are local only to the function in which they were declared and therefore can only be referenced and modified from within said function.

**Note**: BROOM uses static scoping, as does Java, the language into which we are translating.

#### 3.2.3.2 Variable Assignments

Once a variable has been declared, it must be assigned before it can be used. A variable assignment consists of a destination variable, followed by the '=' operator (which denotes assignment) followed by an expression, which the programmer needs to ensure returns a value whose type matches the destination variable, or an error will occur.

In every case save one, assignments will be done by value (i.e. the value returned by the expression will replace the value currently stored in the variable). The exception case occurs when one `Matrix` variable is assigned to another `Matrix` variable (e.g. MyMat = YourMat;). In this case the variable is assigned by reference.

Hence, an assignment can be formally defined as follows:

*ASSIGNMENT : IDENTIFIIER '=' EXPRESSION ;*

Examples of Variable Assignments:

```
A = 1+2;
B = det(c)* (rank(a) - 5);
E = d;
```

**Note:** The terminating semi-colons are not part of the assignment rule, they are part of the statement rule. The semi-colons are included here to make the examples appear more intuitive to those who use the language.

### 3.2.4 Arithmetic Operators

BROOM supports several fundamental operators to be used in expressions. These are the arithmetic operators:

**+ - * /**

Each operator has a different function given the types of its arguments. These behaviors and differences are described below.

#### 3.2.4.1 Addition Operator : +

| Argument 1 Type | Argument 2 Type | Return Type or Error |
|---|---|---|
| `Matrix` | `Matrix` | `Matrix` (see note 1) |
| `Number` | `Number` | `Number` |
| `Number` | `Matrix` | `Error` |
| `Matrix` | `Number` | `Error` |

**Note 1**: Matrix addition can only be performed if each argument Matrix is of the same size. The result will be a Matrix of the same dimension whose elements are the sum of the respective elements in the argument Matrix's. (For more information, see Gilbert Strang's Introduction To Linear Algebra). While the grammar cannot guarantee the dimension equality condition, Static Semantic Analysis will ensure that it is fulfilled.

#### 3.2.4.2 Subtraction Operator : -

This operator performs exactly as the addition operator except the operation is subtraction. See section 2.4.1

#### 3.2.4.3 Multiplication Operator : *

The multiplication operator is the most overloaded basic operator in BROOM. Depending upon its arguments, it can perform scalar multiplication of scalars or matrices, and Matrix multiplication. The table below describes these behaviors.

| Argument 1 Type | Argument 2 Type | Return Type or Error |
|---|---|---|
| `Matrix` | `Matrix` | `Matrix` (see note 1) |
| `Number` | `Number` | `Number` |
| `Number` | `Matrix` | `Matrix` (see note 2) |
| `Matrix` | `Number` | `Matrix` (see note 2) |

**Note 1: `Matrix-Matrix` Multiplication**. Two matrices of dimensions (m1 x n1) and (m2 x n2) can only be multiplied together if n1 = m2. This cannot be enforced by grammar, but if this condition is not met, errors will be thrown to notify users.

**Note 2: `Matrix-Number` Multiplication**. In these two cases, you multiply a `Matrix` with a `Number`. The result is a `Matrix`, of the same dimension as the argument `Matrix`, with all of its `Number` elements multiplied by that scalar.

#### 3.2.4.4 Division Operator : /

The division operator in BROOM is defined to work only on Numbers. Therefore, its argument table is a simple row as follows:

| Argument 1 Type | Argument 2 Type | Return Type or Error |
|---|---|---|
| Number | Number | Number |

**Note**: as in all arithmetic, division by 0 is not permissible and will throw an error.

#### 3.2.4.5 Precedence of arithmetic operators

The precedence of arithmetic operators is the same as in all other arithmetic grammars. Multiplication and Division have the highest precedence and are therefore lower in the AST and evaluated first. Subtraction and Addition are of lower precedence and are therefore higher in the tree and evaluated after Multiplication and Division. Parentheses can be included to change the order of evaluation. Arithmetic expressions in parenthesis are of higher precedence. For more information, see language's precedence order list at end of section 2.5.1.

### 3.2.5 Expressions

Before we begin the formal definition of a statement, it is necessary to define expressions because they will occur frequently within statements. *An expression is a set of optionally parenthesized and/or nested sequences of numbers, matrices, and functions (which return numbers or matrices) connected in sequence with the arithmetic operators.* Expressions take into account the precedence order of the language. This can be seen in the grammar for expressions below:

```
    expr
: term ( (ADD^|SUB^) expr )?
     ;

    term
     : atom ( (MUL^|DIV^) term )?
     ;

    atom
     : funcall
     | builtin_funcall
     | ID
     | signedDouble
     | LPAREN! expr RPAREN!
     | matrix
     ;
```

**Note 1**: funcall, is a function call of a user defined function, for more information see Section 2.7.

**Note 2**: builtin_funcall is a call to a function that is built into the BROOM language. See section 2.8 for more information.

#### 3.2.5.1 Expression Evaluation Precedence List

From these grammar rules, one can discern the precedence level of expressions in BROOM to be as following, in descending order of precedence of evaluation:

1) Function Calls, Built-In Function Calls, Variable, constant `Numbers`, Expressions nested within parentheses, constant `Matrix`'s.
2) Multiplication or Division
3) Addition or Subtraction

### 3.2.6 Statements and Statement Lists

Statements are the basic instructions that comprise most of the body of a given function. A statement can be thought of as a command that you want the program to execute for you. A programming "sentence" as it were. Every function has a statement list (*stmtlist*) which is a list of statements that should be executed in order. A statement list can be thought of as a paragraph in a program. The primary purposes of statements, or the basic instructions that are allowed in the body of a function are:

- **assignments**
- **control-flow**
- **calling a function by itself** (i.e. no return value)
- **returning a value** and escaping the current function scope (if function returns a value)

Hence, the grammar for statements and statement lists naturally follow as:

```
stmtList: (stmt)+
      ;
stmt : assignment | 'return' expr SEMI | control_flow | funcall SEMI
      |printcall SEMI;
```

Assignments have already been treated in a previous section. The next few sections define the *return expressions, control_flow, print calls,* and *function calls* that are considered statements.

#### 3.2.6.1 Conditionals

Conditionals play a major role in the *control_flow* blocks and are therefore treated before the *control_flow* rule. Conditionals ultimately return a boolean value of *'true'* or *'false'*. Their usage is restricted to being place within a conditional clause of a `for`, `while`, or `if` statement conditional.

A conditional is a set of two expressions compared with logical operators, ( expr1 > expr2 ), a conditional preceded by a not operator (!), or a set of 2 conditionals, enclosed in parentheses connected with boolean operators.

Examples of Conditionals:

```
a > b
(a > b) && (b <c)
```

```
!(a <c)
a == 2
b != a
!(a==b)||(12>det(myMatrix - yourMatrix))
```

The && is the logical AND operation, || is the logical OR operation, and '!' is the NOT operation, which are binary operators that operate only on boolean values returned by two expressions connected with a *logical operator*:

| | | |
|---|---|---|
| == | (*LOGEQ*) | - test Number-Number or Matrix-Matrix *equality* |
| < | (*LOGLT*) | - test Number–Number *less-than* |
| > | (*LOGGT*) | - test Number–Number *greater-than* |
| != | (*LOGNEQ*) | - test Number–Number or Matrix–Matrix *inequality* |

The grammar rule that formally defines a conditional follows below:

```
conditional : (logicalExpr1 ((AND^|OR^) logicalExpr1)? )
        ;

logicalExpr1 : (NOT^ LPAREN! logicalExpr RPAREN!) | logicalExpr
        ;

logicalExpr : (expr (LOGGT^|LOGLT^|LOGEQ^|LOGNEQ^) expr )
        ;
```

### 3.2.6.2 Control-flow unit

BROOM supports 3 built-in control flow statements: if, while, and for. The functionality of these control loops is standard:

If-Statements:

```
ifstmt:  'if' '(' conditional ')' 'then'
          stmt-list
          ('else' stmt-list)? 'endif'
```

If the conditional is evaluated as 'true', then the first statement-list is executed. If the conditional is evaluated as 'false', and the optional *'else'* followed by a *stmtlist* is provided, the second statement list is evaluated. If the optional '*else'* and *stmtlist* are not provided, then the if statement is exited. Note that the whole statement must begin with 'if' and end with 'endif'.

While-Statements:

```
whilestmt:  'while' '(' conditional ')'
          stmt-list
          'endwhile'
```

The basic control flow of the while loop is as follows. The parenthesized conditional is evaluated. If it evaluates to true, the enclosed statement list is executed. Once this has been executed, the condition is checked again and the process is repeated until the

condition is evaluated to false. This allows for dynamic iteration but also provides the dangerous possibility of infinite looping.

For-Statements:

```
forstmt: "for" `(` assignment conditional SEMI assignment `)`
          stmtList
            "endfor"!
```

At the beginning of the for statement's execution, the first assignment within the parentheses is executed (note: the variable must be declared outside the for statement and at the beginning of the desired scope). The conditional is then evaluated and if true, the statement list is executed. Once the statement list has finished its execution, the second assignment is executed. Then the condition is checked again and the process continues until the conditional is evaluated to false. Briefly then: the for loop permits an assignment statement that is run once at the very beginning of the loop, a conditional that is checked after each loop iteration, and a iteration statement that is executed at the end of each loop iteration. The assignments and conditions are usually used to implement counters.

### 3.2.6.3 Lone Function Call

Whether or not a function returns a value, it can be called by itself for the benefit of side-effects the function may have. The return value will be discarded. The user also has the ability to define functions that have a 'void' return type. This means that they do not return any value. Therefore, these functions can be called without an assignment. *Print Calls* can be considered lone function calls since they have no return type, but they are defined separately since they have a unique argument type.

### 3.2.6.4 Return Statements

All statements will be defined within a function, whether the function be user-defined or the necessary start function. When a user defines a function, he/she defines what type of argument that function will return (`void, Matrix, Number`). Therefore, inside of a user defined function with a non-void return type, there must a return statement that returns a variable or value that is consistent with the return type of the function. (For more information on functions, see section 2.7) A return statement is defined simply below:

*'return' expr ;*

### 3.2.6.5 Print Calls

BROOM provides the ability to print text, `Matrix's`, or `Numbers` to standard out. There are 2 main built-in functions that are used to print these values to the screen. The first prints text and is defined below:

```
printtext: 'print' `(` TEXT `)`
        ;
TEXT: '"'(~('\n' | '"'))*'"'
        ;
```

Here text is defined as any set of characters without a newline or quotation mark. The given text will be printed out with a terminating newline.
Example of text printing:

```
print("Hello my name is Chris 123")
```

yields in stdout:

```
Hello my name is Chris 123
$ . . .
```

The second type of printing prints a `Matrix` or `Number` to standard out. This is the only call in BROOM that is "postfix". Any `Number` or `Matrix` that is to be printed must be assigned to a variable. Then print is called on that variable according to the following rule:

```
printvariablevalue: ID DOT "print" '(' ')'
      ;
```

If the variable contains a `Number`, the `Number` will be printed without a newline. If the variable contains a `Matrix`, the `Matrix` will be printed to stdout in a readable format.
Example of `Matrix` and `Number` Print Calls:

```
a.print();
b.print();
```

(where a is a `Matrix` and b is a `Number`)

yields in stdout:

```
1  2  3  4
5  6  7  8

4.75$
```

### 3.2.7 Functions

Functions are constructs which, given zero or more parameters, run a series of instructions (statements) and return a type (if return type is not void). They allow for the construction of local variables for usage within the function. Side-effects are defined as changes that a function makes to global variables. Parameters are passed by value and therefore a function cannot indirectly change the value associated with one of its parameters (such an operation would require an assignment). A copy of the parameter's value is created when the function is called so that these values cannot be changed in the instructions within the function. Recursion is supported. Functions can make function calls to themselves.

Every function must be defined according to the following rule:

```
functiondefinition:
        "define" "function" ("Matrix"|"Number"|"Void") ID arglist
        (declaration)*
        (stmtList)
```

```
        "endfunction"
      ;
arglist: `(` (argdecl (COMMA! argdecl)* )? `)'
        ;

argdecl: ("Matrix"|"Number") ID
        ;
```

The "define" and "function" indicate that a function is being defined. The next necessary token is the return type. A function can be defined to return a `Matrix`, `Number`, or nothing (`Void`). The next necessary token is the Identifier which is the name that the function will be referred to as for the duration of the program. Following the function name, a parenthesized list of argument aliases and types are passed. This defines the number of arguments and types that a function can be called on.

The function definition body consists of two sections. First, there is a list of local variable declarations that follow the same pattern as all declarations. Second, there is a list of statements that will be executed when the function is called. Those Identifiers used in the argument list to define the function's arguments can be used in the statements of the function to represent the arguments given on a function call.
**Note**: the power of user defined functions is that they can be called within other functions, allowing the user to encapsulate and structure the flow of their program and allow for more power.

#### 3.2.7.1 An Example of A User Defined Function

```
define function Number giveSum( Number a, Number b, Number c)
       Number sum;

       sum =  a + b + c;

       return sum;
endfunction
```

This simple function has a `Number` return type, takes 3 `Number` arguments and returns their sum. This illustrates the general format of a function definition in BROOM.

#### 3.2.7.2 The Start Function

Every BROOM program must have defined in it a start function. This is analogous to Java and C's 'main' function. This is the actual function that will be called when the program is executed. It has a structure very similar to that of a user-defined function and is defined as follows:

```
startfunction: "start" `(` `)'
        (declaration)*
        stmtList
        "endstart"
```

The main difference between a user defined function and the start function is that the start function definition begins with the word 'start' followed by empty parenthesis. It also has no return statements.

### 3.2.7.3 An Example of A Start Function Definition

```
start()

      Matrix mymatrix,m;
      Number z;

      mymatrix = [ 1, 2, 3, 1-1;
               3, 4, -5, 4;];

      z = sum(1,2,3);

endstart
```

Notice that the start function actually calls the sum function we defined earlier. This is the code that will be executed when the program is run.

## 3.2.8 Built-in functions:

BROOM provides several built-in functions that are inherent to `Matrix` manipulation. The following is a list of those functions that are built in. As can be seen from the syntax, all built-in functions are invoked in a "pre-fix" manner.

1) **`Matrix newMatrix(Number m1, Number n1)`**
   creates a new empty `Matrix`, filled with zeros.
   *m1* specifies the height of this new `Matrix`, and *n1* denotes the height.
2) **`Matrix inv(Matrix a)`**
   computes the inverse of input `Matrix` *a*. *a* must be a square `Matrix` for this function to work properly
   returns the inverse `Matrix`
3) **`Matrix gauss(Matrix a, Matrix b)`**
   Solves the `Matrix` equation $ax=b$, where input `Matrix` *a* is a square `Matrix` and *b* is a vector of the same height as *a*.
   returns the solution x, a column vector of the same height as *a*.
4) **`Matrix trans(Matrix a)`**
   returns the transpose of `Matrix` *a*.
5) **`Number det(Matrix a)`**
   returns the determinant of a `Matrix` *a*.
6) **`Number getElement(Matrix a, Number b1, Number c1)`**
   returns the [*b1*,*c1*]th entry in `Matrix` *a* as a single `Number`.
7) **`Number numCols(Matrix a)`**
   returns the number of columns in `Matrix` *a*.
8) **`Number numRows(Matrix a)`**

returns the number of rows in `Matrix` *a*.

**9) Matrix getRow(Matrix a, Number b1)**

returns a `Matrix` consisting of the b1'th row of `Matrix` *a*.

**10) Matrix getCol(Matrix a, Number b1)**

returns a `Matrix` consisting of the b1'th column of `Matrix` *a*.

**11) Matrix rowSwap(Matrix a, Number b1, Number c1)**

returns a new `Matrix` identical to the `Matrix` *a*, except that the *b1th* and *c1th* rows are swapped.

**12) Matrix colSwap(Matrix a,float b1, float c1)**

returns a new `Matrix` identical to the `Matrix` *a*, except that the *b1th* and *c1th* columns are swapped.

**13) Number rank(Matrix a)**

returns the rank of input `Matrix` *a*.

**14)Matrix setElement(Matrix a,Number b,Number m1,Number n1**)

returns a new `Matrix`, the same as input `Matrix` *a*, with the [*m1*,*n1*]th entry in `Matrix` *a* now set to be equal to *b*.

**15) Matrix setRow(Matrix a, Matrix b , Number pos1)**

returns a new `Matrix`, the same as input `Matrix` *a*, with the *pos1'th* row in `Matrix` *a* now set to be equal to the row `Matrix` *b*.

**16) Matrix setCol(Matrix a, Matrix b , Number pos1)**

returns a new `Matrix`, the same as input `Matrix` *a*, with the *pos1'th* column in `Matrix` *a* now set to be equal to the column `Matrix` *b*.

**17) Matrix addRow(Matrix a, Matrix b)**

returns a new `Matrix`, the same as input `Matrix` *a*, with the row `Matrix` *b* added on as the last row. (height of returned `Matrix` is 1 more than the height of input `Matrix` *a*).

**18) Matrix addCol(Matrix a, Matrix b)**

returns a new `Matrix`, the same as input `Matrix` *a*, with the column `Matrix` *b* added on as the last column. (Width of returned `Matrix` is 1 more than the width of input `Matrix` *a*.

**19) Matrix deleteRow(Matrix a, Number pos1)**

returns a new `Matrix`, the same as `Matrix` *a*, except that row at position pos1 is deleted and the rows under it shifted up by 1.

**20) Matrix deleteCol(Matrix a, Number pos1)**

returns a new `Matrix`, the same as `Matrix` *a*, except that column at position pos1 is deleted and the columns under it shifted left by 1.

**21) Matrix appendLR(Matrix a, Matrix b)**

returns a new `Matrix`, constructed by appending `Matrix` *b* to the right of `Matrix` *a*.

**22) Matrix appendUD(Matrix a, Matrix b)**

returns a new `Matrix`, constructed by appending `Matrix` *b* to the bottom of `Matrix` *a*.

# Chapter 4

# Project Plan

## *4.1 General Project Processes*

The group as a whole followed these general processes in the various stages of the development of the project.

**Planning and Specification**- General guidelines and specifications for the language and the development process were laid out in a number of full group meetings. These meetings were very helpful in order to focus the scope of our language and to determine the most efficient way to implement our goal. Specifications of the interfaces between 2 components were established in these meetings as well.

**Development and Testing** – In general we felt that it was always better to employ "Extreme Programming". Code usually came out more streamlined and less error-prone. However it was difficult to get 2 developers together for extended periods of time. Therefore each was responsible for the individual testing of their components as they developed them. This helped immensely during the final testing stage because those who had developed the components had identified and addressed many errors that would have confused the final debugging process.

## *4.2 Programming Style Guide*

It was generally agreed by the entire team that clear programming style was essential to this project because there were so many interdependent parts. The developer who was creating the Static Semantic Checker needed to have a clearly written grammar with AST specifications commented and explained well so that he could implement the Tree Walker. This clear programming style also allowed for those who did not write a component to change a component without having to wait for the component's author to discern where the error was. This enabled much faster development. The following rules were stated to ease the development process:

1) All code should be clear and well-commented
2) All changes to code should have a comment saying on what date the code was changed and what the reason for change was
3) RCS logs should be updated as well according to rule 2
4) If you update or modify code that you are not responsible for, or that will have an effect on another developer's work (e.g. change in grammar rules or AST composition) you should immediately notify all affected developers.

5) Names should be consistent throughout all programs that deal with the AST and the grammar. Therefore TreeWalker rules should match closely in name to the rules that generated that tree.
6) All Java code, be it back-end or embedded, should have variable and function names that are intuitive and not generic (e.g. float a1,a2,a3 . . )

In general, these rules were followed, did ease development, and cut down on confusion and difficulty in interpreting and/or modifying another developer's code.

## *4.3 Project Timeline*

Below follows the timeline that the team created upon the creation of the white paper.

| **Goal** | **Due Date** |
|---|---|
| *Complete White Paper* | 2-18-03 |
| *Complete Grammar Specifications* | 3-10-03 |
| *Complete Language Specifications and Functions* | 3-10-03 |
| *Language Reference Manual* | 3-27-03 |
| *Final Version of Grammar* | 3-27-03 |
| *Lexer and Parser Complete* | 4-1-03 |
| *Semantic Analysis Complete* | 4-8-03 |
| *Backend Matrix Code Complete* | 4-8-03 |
| *Code Generation Complete* | 4-15-03 |
| *Compiler Assembled Begin Final Testing* | 4-22-03 |
| *Freeze Code and Present* | 4-29-03 |

## *4.4 Team Responsibilities*

It was expected that everyone in the group would help in documentation and would help each of the people below with their responsibilities. The table below indicates what each person actually was responsible for and completed. All helped in testing their various parts during development.

| **Group Member** | **Responsibilities** |
|---|---|
| Michael Weiss | *Static Semantic Analysis, Testing* |
| Chris Tobin | *Grammar(AST), Lexer, Parser, Code Generation* |
| Gabe Glaser | *Back-End Java Matrix Code* |
| Brian Pellegrini | *Testing* |

**Note**: Documentation was generally a group effort. The Final Report was written by Chris Tobin. Chapter 6 was written by Mike Weiss and Brian Pellegrini. Chapter 3 was written by Chris Tobin and Gabriel Glaser.

## *4.5 Software Development Environment*

We did all of our development on Mike Weiss' dual Athlon Processor Machine running Linux. Each group member was given an account on his machine so that they could ssh in and develop. RCS was used for version control. The Lexer, Parser, AST, Semantic Checker and Code Generator were all implemented using the ANTLR language tool with Java 1.4.1. The back-end Java code was written in Java 1.4.1 as well.

## *4.6 Project Log*

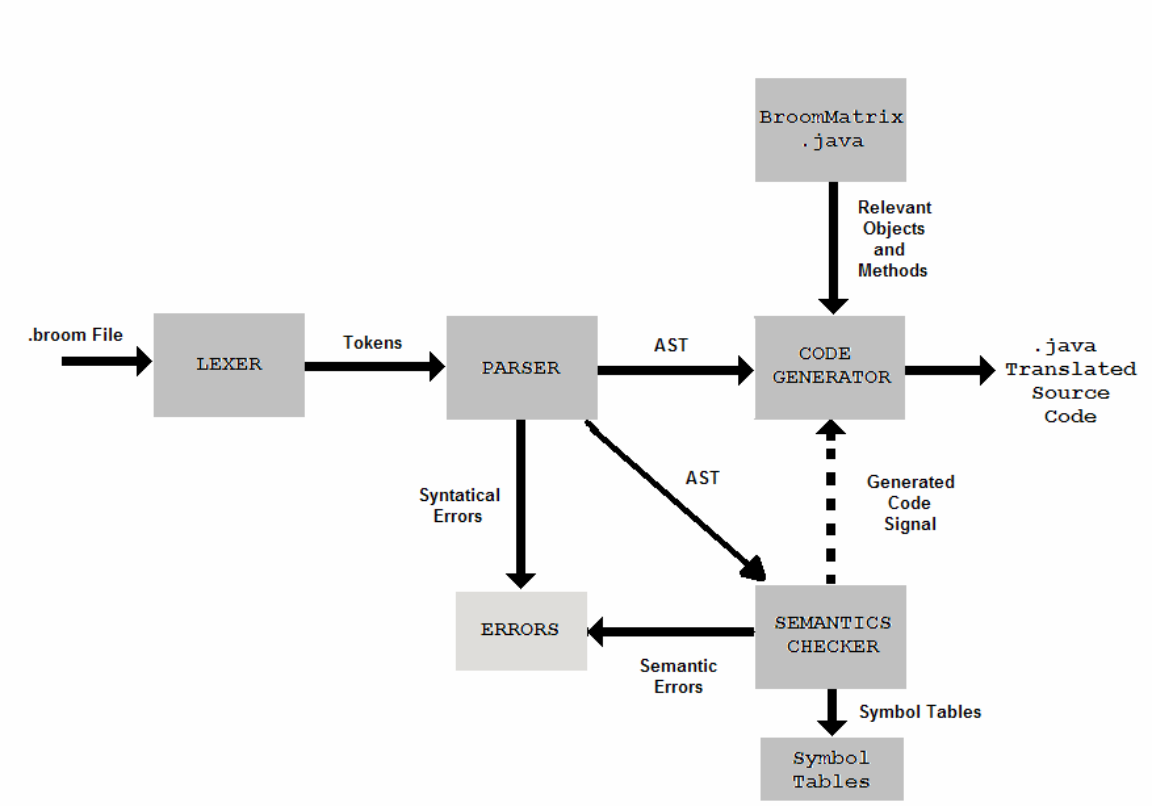
(also see printed RCS logs of the various components at end of this document)

| **Achievement** | **Date Done** |
|---|---|
| *White Paper Completed* | 2-18-03 |
| *RCS and Development Environment Created* | 3-10-03 |
| *Grammar Rough Draft I* | 3-18-03 |
| *Language Reference Manual Completed* | 3-27-03 |
| *Grammar Rough Draft II* | 3-27-03 |
| *Grammar Final Draft* | 4-10-03 |
| *Lexer and Parser Completed* | 4-10-03 |
| *Backend Matrix Code Complete* | 4-10-03 |
| *AST Final Structure Completed* | 4-13-03 |
| *Code Generation Version 1 Completed* | 4-23-03 |
| *Static Semantic Analysis Version 1 Completed* | 4-25-03 |
| *Static Semantic Analysis Completed* | 4-28-03 |
| *Code Generation Completed* | 4-28-03 |
| *Translator Assembled, Final Testing Begun* | 5-4-03 |
| *Documentation Completed/Edited* | 5-9-03 |
| *Code Freeze, Decision To Complete* | 5-9-03 |

# Chapter 5

# Architectural Design of BROOM

## *5.1 Block Diagram of Translator*




## *5.2 Description of Architecture*

There are five main components of the BROOM translator: *lexer*, *parser*, *Semantics Checker*, *Code Generator*, *back-end Java source code library* (`BroomMatrix.java`. A user inputs a .broom file to the translator. The first thing that the translator does is create a *lexer* on that input file. The *lexer* was created using ANTLR and produces the tokens that were defined in the grammar file. This *lexer* is then passed to the *parser* which was also defined in the ANTLR grammar file. The *parser* receives tokens from the *lexer* as is necessary as it parses the file to match the file to the grammar rules that we have defined. If there are syntactical errors, the *parser* reports these errors to the user via standard out. These errors were modified slightly in the grammar so that they would be easier to interpret by the user. If the inputted source file is syntactically correct, the *parser* creates an *Abstract Syntax Tree* or *AST* from the grammar rules. This *AST* is the

main data structure that is used to both test the semantics of the input file and to generate the code.

Once the *AST* has been created, it is passed to the *Static Semantics Analyzer*, or *Semantics Checker*. This is an ANTLR implemented *TreeWalker* that walks the *AST* given by the *parser*. The *Semantics Checker* makes two passes over the *AST*. In the first pass, the *Semantics Checker* creates the appropriate symbol tables for each scope. There are two main scopes in a BROOM program: the global scope (global variables) and the functional local scope (variables local to the function). The *Semantics Checker* creates a symbol table for the global scope and for each defined function. On the second pass, the *Semantics Checker* uses the symbol table and its knowledge of the built-in functions to check the semantics of the inputted program. Some of the many things that the *Semantics Checker* checks are: type-checking, appropriate arguments to functions, no use of unassigned variables, correct argument structure (e.g. taking the inverse of a hard-coded, non-square `Matrix`). If the *Semantics Checker* detects any errors, it reports these errors to standard out. (**Note**: Much time has been invested in trying to create our own *AST* node class based on the ANTLR interface. However, due to time constraints and errors that were difficult to debug, we abandoned this effort. Therefore, the *Semantics Checker* errors are not able to return the line numbers of the errors. Had we had more time, we would have implemented this right away)

The *Semantics Checker* returns a *Vector* of Strings that contains the error messages that it needs to send to the user. If this Vector is empty, that is the "Generate Code Signal" that says that the *AST* is semantically correct and therefore can be used to generate code off of. (The empty error Vector is the dotted-line between the *Semantics Checker* and the Code Generator, it is the go ahead signal to the generator). The static semantic checker examines the following: duplicate variable definitions, duplicate function definitions, invalid assignments, missing return statements, mismatched return types, undefined variables or functions, expression checking, function argument checking, logical expression checking, built-in function checking, and java reserved word testing.

Once the *AST* has been approved by the *Semantics Checker*, the *Code Generator* begins its walk of the *AST*. The *Code Generator* is also an ANTLR created *TreeWalker* that maintains a global String that it accumulates over its walk of the *AST*. This global string contains references to the Objects and methods of the `BroomMatrix.java` file contains the code necessary to create a `Matrix` and perform the various built-in functions. Once it has reached the end of the *AST*, the *Code Generator* prints out this accumulated program string (which is the appropriately translated java source code) to the file name that is specified at the beginning of the .broom file.

## *5.3 Authors of Components*

The following table lists which group member was responsible for implementing and developing the aforementioned components:

| **Group Member** | **Component(s)** |
|---|---|
| Chris Tobin | *Lexer, Parser, Code Generator* |
| Michael Weiss | *Semantics Checker, Symbol Tables* |
| Gabe Glaser | *BroomMatrix.java* |

# Chapter 6:

# Testing Broom Functionality

## *6.1 Testing Overview*

Program testing is an ongoing progress in almost every software development situation. As such, testing has been an integral part of our program since the first day of development. During each phase of development, program segments were run through intermediate test cases for specific tasks. Each group member involved with programming designed small, specialized test cases for their domain. These separate test categories fall into grammar functionality testing, static semantic checker testing, isolated back-end matrix functionality testing, and finally fully-assembled compiler testing. The latter is by far the most important and constitutes the majority of the test phases. Extra effort was placed on assembling the final set of test cases.

### 6.1.1 Grammar Functionality Testing

While the grammar was in the design phase, 2-3 text files containing characteristic grammar constructs were using to get the primary language parser and tree builder up and running. Since the grammar is the foundation of the rest of the language, smaller errors not detected in this phase were soon detected later on. The test text files in this phase were not complete programs. The program was broken down into it's sub rules, each of which were tested individually. These cases, while syntactically correct, were not necessarily semantically correct. In the end, segments of these programs were incorporated into the final and most important testing phase.

### 6.1.2 Static Semantic Checking

Also based on the programming segments which later evolved into finalized test cases, several intermediate test cases were used to test the functionality of the static semantic checker. Eight separate programs were assembled for the intermediate steps of the static semantic checker. These programs do not function in the final version, but were useful in the process of building the static semantic checker.

### 6.1.3 Matrix Operation Checking

The built-in functions each had small test cases not written in broom, but they served to test the matrix operations. The matrix operations were later testing in the final test suite described below.

**NOTE: there are 2 representative test cases shown here in the testing section. For each type of test in the test suite, there are references to the test files that were used. These other files are located at the end of this document in the code listing.**

## *6.2 Static Semantic Checking and Runtime Error Checking*

The next set of test cases were used to check the Static Semantic checker and the error handling of the runtime errors detected by BroomMatrix.java since some errors were undetectable until runtime.

### 6.2.1 Java Reserved Words

Another source of errors in BROOM is the use of reserved words as variable or function names. One important point is that because BROOM is translated into JAVA code the compiler needs to check for the illegal use of both BROOM, and JAVA keywords. This function tests the error checking for both illegal naming of variables, and functions. The compiler was correctly able to identify the use of the JAVA keywords "throws" as a variable name, and "implements" as a function name. This aspect of error checking is very important for obvious reasons. Needless to say reporting the use of BROOM keywords illegally is necessary in order to compile the code correctly. But what is not so immediately important is that the illegal use of JAVA keywords is equally if not more important. At best non-functioning JAVA code would be created leaving the user confused, and thinking that their code logic is implemented incorrectly. At worst it is conceivable that incorrect code is generated that is able to run. Conceivably, this code could produce results that would be incorrect, which would result in disastrous side-effects. These errors are handled by placing all of the JAVA and BROOM keywords into the initial set of symbol tables, and then during variable and function name binding the names are checked against the pre-existing variables and keywords. If a match is found an error is reported, and the process is terminated. BROOM was successfully able to identify and report these errors.

*Please refer to reservedWord.broom and reservedWord.results.txt (by Brian Pellegrini)*

### 6.2.2 Matrix Assignments

A very basic, yet important type of error is matrix assignment where one of the rows has an inconsistent number of columns listed. From the start it is apparent that no matrix operations can be performed on a misshaped matrix, and at a more basic level no array can be declared to hold a matrix with illegal dimensions. As a result it is imperative that these errors be identified and dealt with. The program "wrongNumCols.broom" tests BROOM's ability to deal with this type of error. BROOM was able to identify and report all errors in the program, and terminate the compilation process without generating any code. The results of this test are located in the file "wrongNumCols.result.txt".

*Please refer to wrongNumCols.broom and wrongNumCols.results.txt (by Brian Pellegrini)*

### 6.2.3 Type Checking

Data types are a crucial part of the BROOM language, thus a large part of the testing focused on the use of data types in BROOMs' built-in and user-defined functions. Data types in this language are restricted to two types' real numbers, and matrices. These two types are in no way interchangeable, which means that it is essential we made sure that type checking worked perfectly.

First we tested to make sure that functions would recognize when the types of their arguments were wrong.
*Please refer to unaryOp.broom and unaryOp.results.txt (by Brian Pellegrini)*
Then type checking in logical operations used in control flow constructs was tested
*Please refer to logicalOpps.broom and logicaOpps.results.txt (by Brian Pellegrini)*
Type checking was also checked for agreement of function return types in user defined functions
*Please refer to returnNonVoid.broom and returnNonVoid.results.txt (by Brian Pellegrini)*
*SetRowCol.broom and setRowCol.results.txt (by Brian Pellegrini)*
*Rowcolswap.broom and rowcolswap.results.txt (by Brian Pellegrini)*

### 6.2.4 Run-Time Errors

One possible source of serious errors is in the run time environment. In BROOM run-time errors consist of out of bounds references to indices, and use of improperly formatted matrices for a given matrix operation. To test run-time errors out of bounds errors were tested in "runTimeTest.broom", and improper format errors were tested in "runTimeTest2.broom". To test the out of bounds error the set element function was used. The function was called on indices, which did not exist. As expected BROOM reported that the indices provided were out of the proper range. The results of this test can be seen at "runTimeTest.results.txt". To test for improperly formatted matrices, two matrices with different heights were used with the gaussian elimination function (which is illegal). BROOM correctly caught this error and printed an error message reporting that the two matrices need to have the same height (number of rows). The results of this test can be seen in the file "runTimeTest2.results.txt". These tests are shown at the end of this section.

## *6.3 The Final Test Cases*

The reliable aspects of the language are supported with a discrete set of test cases. These final test sets are divided into two subsets. The first category is the set of working programs. The other consists of programs containing intentional runtime errors, or errors

that will be picked up by the parser and static semantic tree walker to illustrate the error catching mechanisms of our language.

These test cases are included with the broom documentation. Each output is stored in the primary test cases folder. The results of all of these test cases, along with error messages generated by our compiler, are stored as well. There are many test cases in our test suite. We include two examples which best represent the testing routines of the language.

### 6.3.1 The Simple Statistics Program

The basic statistics program included at the end of this section exemplifies a simple program written in broom. It test the simple of functions, numbers, and expressions in a simple demonstration of the language.

*Please refer to Stat2.broom. (by Michael Weiss)*

### 6.3.2 The LU Factorization Program

Also included below is an LU factorization program written in BROOM. It is not written in the most efficient format in order to test most of the features of the language in an applicative fashion. It represents the power of the language well. Essentially, it performs LU factorization on a matrix. It breaks the matrix into two matrices. The first matrix is in lower triangular form. The second matrix is in upper triangular form. The matrices, when multiplied together form the original matrix from which they were factored. The programs perform the factorization and then include the

This data set includes testing of functions, global variables, static semantic checking, and many other aspects of the language. The complex nature of this program demands a fully functional compiler. The program performs without flaws.

*Please refer to ALU.broom. (by Michael Weiss)*

### 6.3.3 A 'HelloWorld' Program

What good is a programming language without a HelloWorld application? This is included to make sure that programs that are very simple still function as they should.

### 6.3.4 MatrixFun Program

This is a generalized random assortment of functions in the programming language that don't accomplish anything specifically, but do serve as a good basis to test certain aspects that were not tested in other sections.

*Please refer to MatrixFun.broom (by Michael Weiss)*

### 6.3.5 Identity Matrix Program

This programs generates an identity matrix using control flow loops and built-in matrix append functions.

*Please refer Identity.broom (by Michael Weiss)*

### 6.3.6 Control Flow Program

This program tests a few control loops to ensure their functionality.

*Please refer to ControMadness (by Michael Weiss)l*

### 6.3.7 Gaussian Elimination Program

This program tests simple Gaussian elimination in order to test the built-in functions just a little.

*Please refer to Gauss (by Michael Weiss)*

## *6.4 A Final Note on Testing*

No test set is ever fully thorough, as it is extremely difficult to catch all of the possible errors that a programmer may create with his program. However, our test suite, included separately in our programming language package, lays the ground for a set of programs which represent a subset of the languages that a BROOM programmer may encounter.

*Breakdown of test case labor*
*Primary Grammar Debugging and Testing – Chris Tobin and Michael Weiss*
*Static Semantic Checking Cases – Michael Weiss and Brian Pellegrini*
*Runtime Error Checking – Brian Pellegrini*
*Matrix Function Checking – Gabriel Glaser, Michael Weiss and Brian Pellegrini*
*Primary Test Case Suite – Michael Weiss*

**Broom Test Source Code**

```
Statistics
// test case by Michael Weiss
//
// this program computes the statistics
// which are stored in a matrix
//
global Matrix data1,data2;
data1 = [ 1,2,3;
          4,5,6;
          7,8,9; ];
```

```
data2 = [ 0,1,0;
          1,0,1;
          1,1,1; ];
define function Number findAverage( Matrix d )
        Number i,j;
        Number m,n,avg,sum,num;

        m = 1; n = 1;
        m = numRows( d );
        n = numCols( d );

        num = m*n;

        sum = 0;

        for ( i=0; i<m; i=i+1; )
                for (j=0; j<n; j=j+1; )
                        sum = sum + getElement(d,i,j);
                endfor
        endfor
        avg = sum / num;
        return avg;
endfunction

start()
       Number a,b;

        a = findAverage( data1 );
       print("average:");
        a.print();
        print("");

endstart
```

## The Corresponding Java Output : Statistics.java

```
//Thus begins your broom program
public class Statistics
{
BroomMatrix  data1, data2;
public Statistics()
{
        float[][] _temp0 = {{(float) 1, (float) 2, (float) 3}, {(float) 4, (float) 5,
(float) 6}, {(float) 7, (float) 8, (floa
t) 9}};
        data1 =  new BroomMatrix( _temp0);
        float[][] _temp1 = {{(float) 0, (float) 1, (float) 0}, {(float) 1, (float) 0,
(float) 1}, {(float) 1, (float) 1, (floa
t) 1}};
        data2 =  new BroomMatrix( _temp1);
}
public float findAverage ( BroomMatrix d)
{
          d = BroomMatrix.copy(d);
        float  i, j;
        float  m, n, avg, sum, num;
        m = (float) 1;
        n = (float) 1;
        m =  BroomMatrix.numRows(d);
        n =  BroomMatrix.numCols(d);
        num = BroomMatrix.multiply(m,n);
        sum = (float) 0;
        for( i = (float) 0;  i < m;  i = BroomMatrix.add(i,(float) 1)   )
        {
                for(  j = (float) 0;  j < n;   j = BroomMatrix.add(j,(float) 1)   )
                {
                        sum = BroomMatrix.add(sum, BroomMatrix.getElement(d, i, j));
                }
        }
        avg = (float)sum/ (float)num;
```

```
        return avg;
}

public void start()
{
        float  a, b;
        a = findAverage(data1);
        System.out.println("average:");
        BroomMatrix.printVariable(a);
        System.out.println("");
}

public static void main(String[] args)
{       Statistics myProgram = new Statistics();
        myProgram.start();
}
}
```

## Program Output

```
[mjw133@aluminor plt-project]$ java Statistics
average:
5.0
```

---

## *RuntimeTest2*

```
runTimeTest2

// Run Time errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//run-time errors works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1; 2,0;];
B= [0, 1; 1, 0;];


start()
        //matrices have different heights
        //should cause a run time error
        A = gauss(A, B);


Endstart
```

## RunTimeTest2 Output:

```
Run-time test for improperly formatted matrices:

Input matrix (first parameter) to gaussian elimination must be square
B vector (second parameter) to gaussian elimination must be an M x 1 matrix.

0.0 1.0
1.0 0.0
is not M x 1, it has 2 columns.
The height (M) of the B vector must equal the height (M)of the (A) input matrix.
Here, (A) has 3 rows, but (B) has 2 rows.
```

## RunTimeTest.broom Source Code:

```
runTimeTest

// Run Time errors
// Author Brian Pellegrini
```

```
// This is supposed to make sure that error checking for
//run-time errors works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1; 2,0;];
B= [0, 1; 1, 0;];


start()
        //row number is out of bounds
        //should cause a run time error
        A = setElement(A, 5, 1, 10);


Endstart
```

## RunTimeTest.broom Output:

```
Results of run time test for out of bounds error:


Requested N coordinate (10) is out of range. Width range is 0 to 1
setElement: Element requested out of matrix range
```

# Chapter 7

# Lessons Learned

## *7.1 Each Member's Lesson Learned*

**Chris Tobin** – I found that the most important thing in this project was delegation of work, communication and personal responsibility. It was very important that every person know what was going on with each component and when. When this didn't happen, confusion ensued. I would suggest to future groups that they set strict deadlines for themselves. This project can seem deceptively shorter than it is. I would also suggest that every person on the team learn ANTLR syntax so that they can understand what is written by other team members. In terms of the class, I would suggest that the professor have more due dates and deadlines to force the students to adhere to a schedule.

**Michael Weiss** - I found that good planning, organization, caffeine, and hours of coding were essential ingredients in designing a programming language. While writing the static semantic checker, I learned the following: a list of program tasks to implement and version control are essential tools. Good team communication is necessary for project interdependencies. Lastly, thorough testing rolls out all of the lumps in the dough of the program. Regular scheduling of intermediate deadlines greatly aided the flow of the project design.

**Gabe Glaser –** As in any group project, I found that the most important thing as far as the design and implementation process goes is organization. As clichéd as that sounds (obviously it is important), it cannot be overstated as to how important this really is. Assignment of tasks and responsibility, along with intercommunication between group members is what allowed us to use a streamlined, modular development model where each member could build their own piece of the project, working with standards that we have imposed in order to treat each module as a 'black box' component. There is also incentive for each team member to learn about what the other team members are doing so that when the time comes to start the integration phase, there is little room for confusion and repeated work.

**Brian Pellegrini** - I for one learned a whole lot in the process of completing this project. First and foremost I learned that when working on a large scale project with a group it is imperative that there be a game plan. In addition communication and scheduling are key because they ensure that the game plan stays on track. Overall I learned that coordinating, and controlling group efforts is just as important (and sometimes more difficult) as the coding itself.

# Appendix L:
# (The Long appendix)

# *Code Listing:*

```
THE MAKEFILE WE USED:

makefile (mjw133)
-------------------------------------------------------------------------------

.DEFAULT:
	co RCS/$@,v

# makes everything that we have so far (default, just type "make")
all: grammars testingfiles semanticfiles main
	javac *.java

clean:
	rcsclean

main: BCC.java

grammars: broom.treewalker.g broom.test.g codegenerator.g
	java antlr.Tool broom.test.g
	java antlr.Tool broom.treewalker.g
	java antlr.Tool codegenerator.g

testingfiles: Main.java program.test.txt

# these are for the static semantic checker
semanticfiles: StaticMain.java BroomSymbolTable.java Symbol.java Context.java

-------------------------------------------------------------------------------



NOTE: For rlog files: authors: mjw133 -- Michael, cmt & cmt69 Chris Tobin
                               gg - Gabriel Glazer (did most work from home so checkins
                                                    are less frequent)
                               bdp - brian pellegrini

-------------------------------------------------------------------------------
The Compiler - BCC.java   Authors: Michael Weiss, Gabriel Glaser, Chris Tobin
-------------------------------------------------------------------------------

import java.io.*;
import java.util.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;


/**
 *  Translates a broom program into java.
 *  contains the main method.
 */
public class BCC
{

    //please see the string USAGE for information on USAGE
    //
    public static void main(String[] args) {
```

```
        try {
            if ( args != null && args.length == 1 && args[0] != null ) {
                String filename = args[0];

                System.out.println(filename);

                StringTokenizer tokenizer = new StringTokenizer( filename,"." , false );

                String lastToken = "error";
                while ( tokenizer.hasMoreTokens() ) {
                    lastToken = tokenizer.nextToken();
                }


                if ( lastToken.equalsIgnoreCase("broom")  ) {

                    DataInputStream in = new DataInputStream ( new FileInputStream (
filename ));

                    //build the lexer object
                    broomL lexer = new broomL(new DataInputStream(in));

                    //build the parser
                    broomP parser = new broomP(lexer);

                    //parse the file
                    parser.startRule();

                    //get the parse tree
                    CommonAST parseTree = (CommonAST)parser.getAST();

                    //run the static semantic checker
                    StaticSemanticWalker swalker = new StaticSemanticWalker();

                    //run the primary phase of the static checker (global
variable/function) table
                    swalker.buildGlobalTable(parseTree);

                    //run the rigourous static semantic check
                    swalker.semanticCheck(parseTree);

                    //get the vector containuing all of the errors
                    Vector errorVector = swalker.getErrorVector();

                    //run the code generation if there were no errors
                    if ( errorVector == null ) {

                        CodeGenerator cg = new CodeGenerator();

                        //run code generation
                        cg.buildCode( parseTree );

                    } else {

                        //print the semantic errors and quit
                        System.out.println("BroomProgram did not compile; the following
errors were detected:");

                        for ( int i = 0; i < errorVector.size(); i++ ) {
                            String error = (String) errorVector.elementAt(i);
                            System.out.println(error);
                        }

                        //indicate an error occured with 1
                        System.exit(1);
                    }

                } else {
                    System.out.println("Invalid filename.  Expecting *.broom.");
                    printUsageAndExit();
                }
```

```
            } else {
               printUsageAndExit();
            }

            //catch exceptions generated during parsing
        } catch(IOException e) {
            System.out.println("Error occured while reading your broom file.");
            e.printStackTrace();
             System.out.println("exception: "+e);
         } catch(antlr.RecognitionException re) {
            System.out.println("Recognition Exception occured while reading your broom
file.");
            re.printStackTrace();
             System.out.println("exception: "+re);
        } catch(antlr.TokenStreamException re) {
            System.out.println("Token Stream Exception occured while reading your broom
file.");
            re.printStackTrace();
             System.out.println("exception: "+re);
        }
    }

    /** Usage is fairly straightforward.
        Make sure that the file ends in .broom */
    public static final String USAGE = "java BCC [broomfile]";

    /**
     * Prints how to use the program and terminates
     */
    public static void printUsageAndExit() {
        System.out.println(USAGE);
        System.exit(1);
    }

}

----------------------------------------------------------------------------

RCS LOG FILES ATTACHED:

----------------------------
revision 1.4
date: 2003/05/12 18:33:19;  author: mjw133;  state: Exp;  lines: +15 -5
Cleaned the file up for the final turnin.
----------------------------
revision 1.3
date: 2003/05/09 19:42:44;  author: mjw133;  state: Exp;  lines: +1 -1
Fixed help message.
----------------------------
revision 1.2
date: 2003/05/09 17:35:26;  author: mjw133;  state: Exp;  lines: +1 -1
Fixed trivial error.
----------------------------
revision 1.1
date: 2003/05/05 02:22:37;  author: mjw133;  state: Exp;
Initial revision
============================================================================



----------------------------------------------------------------------------
grammar.test.g -- our grammar file, performs the parsing
Author: Chris Tobin
----------------------------------------------------------------------------

{
    import antlr.CommonAST;
}
class broomP extends Parser;

options {
```

```
    buildAST = true;
    k=2;
    exportVocab = broom;

    //this would have been for line numbers, but time did not
    //permit implementation
    //ASTLabelType = "LineAST";
}

tokens {
    BUILTIN_FUNCALL;
    NOARYOP;
    DECLARATION;
    IFSTMT;
    STMTLIST;
    WHILESTMT;
    FUNCTIONDEFINITION;
    ARGLIST;
    ARGDECL;
    START;
    FUNCALL;
    EXPRLIST;
    FORSTMT;
    BINARYOP;
    UNARYOP;
    LOGICALEXPR;
    MATRIX;
    MATRIXROW;
    TERNARYOP;
    PRINTTEXT;
    PRINTVARIABLEVALUE;
}

//5-1-03, cmt69, program now starts with ID which is the name of the program
startRule : ID^ program;

//4-9-03, cmt69, adding in final program structure

program : (globalvariabledec)*
          (globalvariableassignment)*
          (functiondefinition)*
          startfunction
        ;

globalvariabledec : "global"! declaration;

declaration : ("Matrix"|"Number") ID (COMMA! ID)* SEMI!
        { #declaration = #([DECLARATION, "declaration"], declaration);}
        ;

globalvariableassignment : assignment;

functiondefinition: "define"! "function"! ("Matrix"|"Number"|"Void")? ID arglist
        (declaration)*
        (stmtList)
        "endfunction"!

        {#functiondefinition = #([FUNCTIONDEFINITION, "functiondef"],
functiondefinition); }; //for now

//4-12-03, cmt69, adding arglist for function definition
//4-26-03, cmt69, correcting arglist to include commas
arglist: LPAREN! (argdecl (COMMA! argdecl)* )? RPAREN!
        {#arglist = #([ARGLIST, "arglist"], arglist);}
        ;

argdecl: ("Matrix"|"Number") ID
        {#argdecl = #([ARGDECL, "argdecl"], argdecl);}
        ;

startfunction: "start"! LPAREN! RPAREN!
```

```
        (declaration)*
        stmtList
        "endstart"!
        {#startfunction = #([START, "start"], startfunction);}
    ;


//4-9-03, cmt69, adding in statements

stmt : assignment | "return"^ expr SEMI! | funcall SEMI! | control_flow | printtext SEMI!
| printvariablevalue SEMI!
        ;

stmtList: (stmt)+
        {#stmtList =#([STMTLIST, "stmtList"], stmtList);}
        ;

assignment: (ID EQUALS^ expr SEMI!)
        ;
//4-25-03, cmt69, adding in the print statement
printtext: PRINT! LPAREN! TEXT RPAREN!
        {#printtext = #([PRINTTEXT, "printText"], printtext);}
        ;
//4-26-03, cmt69, adding in print statement for matrices and numbers, so that it is not
in the
//builtin_funcall.  the reason for this is that all builtin_funcall's return
//some value.  a.print() does not, so it cant be in that rule.
//realize that this will print print BOTH matrices and numbers, so gabe's rules need to
have a
//print statement for NUMBERS!

printvariablevalue: ID DOT! PRINT LPAREN! RPAREN!
        {#printvariablevalue = #([PRINTVARIABLEVALUE,"printvariablevalue"],
printvariablevalue);}
        ;


//4-9-03, cmt69, adding in Control Flow Constructs

control_flow: ifstmt| whilestmt |forstmt;

ifstmt : "if"! LPAREN! conditional RPAREN! "then"! stmtList
        ("else"! stmtList)? "endif"!
        {#ifstmt = #([IFSTMT, "ifstmt"], ifstmt);}
        ;

whilestmt: "while"! LPAREN! conditional RPAREN!
        stmtList "endwhile"!
        {#whilestmt = #([WHILESTMT, "whilestmt"], whilestmt);}
        ;

//mjw133, fixed semi missing semi-colon in for statement
forstmt: "for"! LPAREN! assignment conditional SEMI! assignment RPAREN! stmtList
        "endfor"!
        {#forstmt = #([FORSTMT, "forstmt"], forstmt);}
        ;


//4-9-03, cmt69, adding in Conditionals

//condtional tree structure looks good as it is
conditional : (logicalExpr1 ((AND^|OR^) logicalExpr1)? )
        ;

logicalExpr1 : (NOT^ LPAREN! logicalExpr RPAREN!) | logicalExpr
        ;

logicalExpr : (expr (LOGGT^|LOGLT^|LOGEQ^|LOGNEQ^) expr )
        {#logicalExpr = #([LOGICALEXPR,"logicalExpr"], logicalExpr);}
        ;
```

```
//below follows rules for expressions that work
//with the help of edwards
expr
  : term ( (ADD^|SUB^) expr )?
  ;

term
 : atom ( (MUL^|DIV^) term )?
 ;

atom
 : funcall
 | builtin_funcall
 | ID
 | signedDouble
 | LPAREN! expr RPAREN!
 | matrix
 ;


//4-12-03, cmt69, adding in matrix syntax for declarations
//AST: matrix node, with row items as its children
//
//4-12-03 causing major non-determinism, cannot resolve
matrix:
        LBRACK! (matrixrow)+ RBRACK!
        {#matrix =#([MATRIX,"matrix"], matrix); }
        ;

//if signed double or id is made into expr, it causes non-determinism
//problem is that we need to have expressions inside the matrix so that you
//can do things like:
//
//[1+2 3 4 5;
// a b c d.inv();];
//
//this should be allowed and we cant get that in the grammar right now.
//
//4-12-03 - got this to work with no non-determinism, only problem is that everything
//needs to be separated by commas.
matrixrow: (expr) (COMMA! expr)*  SEMI!
        {#matrixrow = #([MATRIXROW, "matrixrow"], matrixrow);};


funcall
  :  ID LPAREN! (expr_list)? RPAREN!
        {#funcall = #([FUNCALL,"funcall"],funcall);}
  ;

expr_list : ( expr ( COMMA! expr )* )
        {#expr_list =#([EXPRLIST,"expr_list"], expr_list);}
          ;
//4-27-03, cmt69, this call would not allow for such things as the determinant of the
//inverse of a matrix.  the postfix syntax was stupid looking eg. (a.inv()).det().  so,
even
//though its late in the game, were gonna change the builtin_funcalls to look like normal
//funcalls.  eg.  det(inv(a)).  the old rule remains commented out in case we run into
problems
//and need to revert.  the only nice thing about the old rule was that only one function
coulde
//be called at a time which would simplify the tree.  so if the tree is getting crazy, we
might
//want to change this back.
builtin_funcall

    :
        (noaryOp LPAREN! expr RPAREN!
        | unaryOp LPAREN! expr COMMA! expr RPAREN!
        | binaryOp LPAREN! expr COMMA! expr COMMA! expr RPAREN!
        | ternaryOp LPAREN! expr COMMA! expr COMMA! expr COMMA! expr RPAREN!)
        {#builtin_funcall = #( [BUILTIN_FUNCALL,"builtin_funcall"] , builtin_funcall);}
```

```
  ;


signedDouble : (SUB NUMBER^) | NUMBER^ ;

//the matrix operators
noaryOp : ( INV | DET | TRANS | RANK | NUMROWS | NUMCOLS )
        {#noaryOp = #([NOARYOP, "noaryOp"], noaryOp);}
        ;
unaryOp : ( GAUSS | ROW | COL | ADDROW | ADDCOL | APPENDLR | APPENDUD |DELETEROW |
DELETECOL | NEWMATRIX)
        {#unaryOp = #([UNARYOP, "unaryOp"], unaryOp);}
        ;

binaryOp : ( ELEMENT | ROWSWAP | COLSWAP | SETROW | SETCOL)
        {#binaryOp = #([BINARYOP, "binaryOp"], binaryOp);}
        ;

ternaryOp : (SETELEMENT)
        {#ternaryOp =#([TERNARYOP, "ternaryOp"], ternaryOp);}
        ;


/*-------------------------------------------------------------------------------------
--
LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER
LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER LEXER

Authors: Chris Tobin   Secondary Author to Lexer not grammar: michael weiss
-------------------------------------------------------------------------------------
*/
class broomL extends Lexer;

options {

        k=3;
        exportVocab = broom;
        charVocabulary = '\3'..'\377';
}

tokens {
      //built-in functions
      GAUSS="gauss";
      INV="inv";
      DET="det"; // determinant
      TRANS="trans"; // transpose
      RANK="rank";
      SETELEMENT="setElement";
      SETROW="setRow";
      SETCOL="setCol";
      PRINT = "print";

       //accessor methods
      ROWSWAP="rowSwap"; //rowswap
      COLSWAP="colSwap";
      ROW="getRow"; //row for access
      COL="getCol"; //column for access
      ELEMENT="getElement"; //element
      NUMROWS="numRows"; //m
      NUMCOLS="numCols"; //n
    //5-1-03, cmt69, adding in addrow, addcol and append commands
      ADDROW="addRow";
      ADDCOL="addCol";
      APPENDLR ="appendLR";
      APPENDUD ="appendUD";

    //5-2-03, cmt69, adding in deletefor, and deletecol
      DELETEROW = "deleteRow";
      DELETECOL = "deleteCol";

    //5-2-03, cmt69, adding in the newMatrix method which returns a matrix full of 0's
```

```
      NEWMATRIX = "newMatrix";

}

ID
options {
    paraphrase = "an identifier";
}
    : (('a'..'z')|('A'..'Z')) ( (('a'..'z')|('A'..'Z')) | DIG )*
    ;

TEXT: '"'(~('\n' | '"'))*'"';

NUMBER : DECIMAL ( 'e' (SUB)? DECIMAL )? ;

protected DECIMAL : (DIG)+ (DOT (DIG)+)?;

protected DIG : ('0'..'9');

LPAREN : '(' ;
RPAREN : ')' ;
COMMA  : ',' ;
LBRACK : '[' ;
RBRACK : ']' ;
SEMI   : ';' ;
DOT    : '.' ;


// arithmetic operators
ADD : '+';
SUB : '-';

MUL : '*'; //handles scalar multiplication, m * m, m * s, s * m
DIV : '/'; //handles scalar division


// exponential operator
POW : '^'; //to be implemented

EQUALS
options {
    paraphrase = "an assignment operatior '='";
}
    :'='
    ;

// boolean operators
NOT : '!';
AND : "&&";
OR  : "||";

//logical operator s
LOGEQ : "==" ;
LOGNEQ : "!=" ;
LOGGT : '>' ;
LOGLT : '<' ;

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

//4-25-03, comments are put in
//4-27-03, cmt69, allowing quotes to be in the comment
COMMENTS :
        ('/''/'(~('\n' ))*  '\n')
        {newline();}
        {$setType(Token.SKIP);}
        ;
```

```
total revisions: 23;  selected revisions: 23
description:
An intermediate level for developing grammer.
This one currently handles numbers and has a funny list of numbers
thing.  Will be expanded later on.  Initial Revision.
----------------------------
revision 1.23
date: 2003/05/12 18:35:06;  author: mjw133;  state: Exp;  lines: +2 -1
Updated comments for final revision.
----------------------------
revision 1.22
date: 2003/05/04 23:47:11;  author: cmt;  state: Exp;  lines: +4 -1
changed builtin_funcall to include the newMatrix rule which takes the
# of rows and then the # of cols.
----------------------------
revision 1.21
date: 2003/05/03 20:23:57;  author: mjw133;  state: Exp;  lines: +1 -1
Need to add "Void" return types so that the static semantic
checker can do it's job.
----------------------------
revision 1.20
date: 2003/05/03 20:08:40;  author: mjw133;  state: Exp;  lines: +19 -4
Tried to add special AST nodes that would allow for
line number reporting on the static semantic checker,
but it did not work.  Only changed only line of code
and have remarked changes out.

Also, added to paraphrase's to the LEXER for slightly
better error reporting.
----------------------------
revision 1.19
date: 2003/05/02 19:38:38;  author: cmt;  state: Exp;  lines: +3 -2
added in colswap
----------------------------
revision 1.18
date: 2003/05/02 17:46:58;  author: cmt;  state: Exp;  lines: +5 -21
changed the grammar to take the deleteRow and deleteCol
----------------------------
revision 1.17
date: 2003/05/02 02:52:13;  author: cmt;  state: Exp;  lines: +2 -1
changes start rule so that the first token is an IDENTIFIER which is the desired name of
the program
MIKE, YOU GOTTA CHANGE STATIC SEMANTIC WALKER TO REFLECT THIS CHANGE
----------------------------
revision 1.16
date: 2003/05/01 19:44:54;  author: cmt;  state: Exp;  lines: +7 -1
cmt69, 5-1-03.  added into the builtin_funcall rule the ability
to have the appendUD, appendLR, addRow, addCol function calls.
static semantics needs to worry about this to make sure that it only gets
matrix args.
----------------------------
revision 1.15
date: 2003/04/28 20:51:46;  author: mjw133;  state: Exp;  lines: +1 -1
reverted back to previous version after introducing error.... oops
go rid of semi-color, realizing it was included in the assignment
----------------------------
revision 1.14
date: 2003/04/28 20:43:47;  author: mjw133;  state: Exp;  lines: +2 -1
In the for statement, added a semicolor after the assignment
so that the for body looks like
for ( i=0 ; i<5 ; i=i+1 )
endfor

instead of:
for ( i=0 i<5 ; i=i+1 )
endfor
----------------------------
revision 1.13
date: 2003/04/27 21:31:36;  author: cmt;  state: Exp;  lines: +2 -1
cmt69, changed comment rule to allow quotation marks.
```

```
----------------------------
revision 1.12
date: 2003/04/27 20:16:57;  author: cmt;  state: Exp;  lines: +5 -5
cmt69, 4-27-03, just changed a few of the token strings for the built in function
calls, so that they reflect the function names that gabe used in his java files
----------------------------
revision 1.11
date: 2003/04/27 18:12:37;  author: cmt;  state: Exp;  lines: +25 -8
BUILTIN FUNCALL CHANGED!:
Built in function calls are no longer post fix.  meaning its no longer a.inv().
now it's inv(a).
this allows us to nest builtin function calls.
with the old method this was not possible.  may have to roll back the update if this is
too hard
to do with the tree walker, we might have to simplify this rule to make the compiler
work.
----------------------------
revision 1.10
date: 2003/04/26 20:07:05;  author: cmt;  state: Exp;  lines: +1 -0
were at 1.10, madness!
small change to the grammar for the conditionals.  now you can use ! on a
logical expression.  so you can write !(a ==b).  that works now.  remember though
the logical expression that is being notted needs to be nested in parentheses.
----------------------------
revision 1.9
date: 2003/04/26 20:03:02;  author: cmt;  state: Exp;  lines: +5 -6
cmt69, 4-26-03, the grammar now works with the conditionals
----------------------------
revision 1.8
date: 2003/04/26 17:14:33;  author: cmt;  state: Exp;  lines: +15 -3
cmt69, function definitions now recoginize appropriate arglist
a.print()'s are now void methods, meaning that they return nothing and are
not a part of the builtin_funcall rule.
note:  these a.print()'s now apply to all variables, be they matrices or numbers
this means that in gabe's code, we need to have something that will have a
similar print method for our numbers.  might be a problem.
----------------------------
revision 1.7
date: 2003/04/25 17:49:20;  author: cmt;  state: Exp;  lines: +27 -4
This grammar now has a print method
print("hello my name i sjo");
strings inside cannot contain quotes or newlines
also has matrix print:
a.print();
also added in comments with //
----------------------------
revision 1.6
date: 2003/04/12 21:27:54;  author: cmt;  state: Exp;  lines: +4 -1
non-determinism in matrix has been solved, but now we have to have the elements of each
row seperated by commas.
----------------------------
revision 1.5
date: 2003/04/12 19:48:29;  author: cmt;  state: Exp;  lines: +107 -30
this now creates the AST well.  there is one MAJOR problem, and htat is that
the matrix rule is not working.  it wont allow us to nest expressions inside
of the matrix. e.g. [1+2 3;]; is not allowed bc 1+2 is an expression
and putting the expression rule into the matrix rule causes non-determinism
were gonna have to look hard at this, and if that doesnt work go to edwards.
right now the matrix rule works by accepting only variables and signed
doubles.  it works fine like that and the AST looks good.  need to talk to group about
non-determinism and problem
----------------------------
revision 1.4
date: 2003/04/10 04:01:26;  author: cmt;  state: Exp;  lines: +104 -86
everything is working except:
need to add matrix rule for our matrix definitions (i,e, [1 2 3; 4 5 6;];)
also need to decide about function definitions: should we seperate declaraion and
definition?
finally, maybe we should make built-in function names the nodes of the builtin calls
----------------------------
revision 1.3
```

```
date: 2003/04/09 04:18:46;  author: cmt;  state: Exp;  lines: +5 -4
problem has not been resolved from 1.2
----------------------------
revision 1.2
date: 2003/04/09 01:48:09;  author: cmt;  state: Exp;  lines: +70 -7
This now has the expression rule working except that terms cannot be surrounded by
parens.  ie.  the following would fail:
myFun(1, 2, 3 * (a))
this is because the (a) following the * is meant to be a term which so far cannot be in
parens.  so this needs to change
other than that, expressions kick ass.
----------------------------
revision 1.1
date: 2003/04/08 21:20:00;  author: cmt;  state: Exp;
Initial revision
=============================================================================
```

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

//static semantic checker

// all code and documenation:
// -- by Michael Weiss

//this class keeps track of the errors in a vector
//after running first the global table builder
//then the static checker function, if the vector
//is not null, then there are errors, contained
//as strings in the vector
//
//print these and do not generate code from these
//errors
//
//NOTE: if the static semantic checker is given
//a file that is syntactically incorrect, then
//it will not function properly.
//
//proper error messages appear when the file is syntactically
//correct, or least mostly, but contains only semantic errors
//
//on syntactically incorrect files, some errors may be detected
//but there is no garuntee for completeness
class StaticSemanticWalker extends TreeParser;
options {
    importVocab = broom;
}

{
    //indicates whether or not to display error messages
    private boolean debugMode = false;

    private boolean globalScope = false;

    //our global symbol table
    public BroomSymbolTable globalTable;

    //our local context and global symbol table together
    //creating a context for one function
    public Context context;

    //for error messages -- indicating the location
    public String curFunc = "*global declaration*";

    //keeps track of the expected return type for a function
    //in case the wrong type is returned.
    public int expectedReturnType = Symbol.VOID;
```

```
    // check if m==n for matrix dimension checking
    public boolean isSquare( int[] dimensions ) {
        return ( dimensions[0] == dimensions[1] );
    }

    //java reserved words && builtin class words cannot be used...build a hash table
    //which contains these words
    public Hashtable rwords;

    // contains all of the errors that the static semantic checker picks up
    // (as strings)
    Vector ev;

    // returns the error vector
    public Vector getErrorVector() {
        return ev;
    }

    //adds an error to the error vector in the proper
    //format, also recording the location
    public void err( String errMsg ) {
        if (ev == null ) { ev = new Vector(); }

        String error = new String();
        error += "* Error in " + curFunc + " : " + errMsg;
        //System.out.println(error);
        ev.add( error );
    }

    //a lengthy method to build the hash table of java reserved keywords
    //I only include it here to avoid any file dependencies
    public void buildrtable() {

        rwords = new Hashtable();

        // code below generated by 1-liner awk program: { printf
"rwords.put(\"%s\",\"%s\");\n", $0, $0 } from rwords.txt
        // sorry for the generated code right upfront
        // - hand writen code is below, but this generated code avoids any extra file
dependencies on runtime.
        rwords.put("abstract","abstract");
        rwords.put("default","default");
        rwords.put("if","if");
        rwords.put("private","private");
        rwords.put("this","this");
        rwords.put("boolean","boolean");
        rwords.put("do","do");
        rwords.put("implements","implements");
        rwords.put("protected","protected");
        rwords.put("throw","throw");
        rwords.put("break","break");
        rwords.put("double","double");
        rwords.put("import","import");
        rwords.put("public","public");
        rwords.put("throws","throws");
        rwords.put("byte","byte");
        rwords.put("else","else");
        rwords.put("instanceof","instanceof");
        rwords.put("return","return");
        rwords.put("transient","transient");
        rwords.put("case","case");
        rwords.put("extends","extends");
        rwords.put("int","int");
        rwords.put("short","short");
        rwords.put("try","try");
        rwords.put("catch","catch");
        rwords.put("final","final");
        rwords.put("interface","interface");
        rwords.put("static","static");
        rwords.put("void","void");
```

```
        rwords.put("char","char");
        rwords.put("finally","finally");
        rwords.put("long","long");
        rwords.put("strictfp","strictfp");
        rwords.put("volatile","volatile");
        rwords.put("class","class");
        rwords.put("float","float");
        rwords.put("native","native");
        rwords.put("super","super");
        rwords.put("while","while");
        rwords.put("const","const");
        rwords.put("for","for");
        rwords.put("new","new");
        rwords.put("switch","switch");
        rwords.put("continue","continue");
        rwords.put("goto","goto");
        rwords.put("package","package");
        rwords.put("synchronized","synchronized");
        rwords.put("Array","Array");
        rwords.put("Byte","Byte");
        rwords.put("Character","Character");
        rwords.put("Character.Subset","Character.Subset");
        rwords.put("Character.UnicodeBlock","Character.UnicodeBlock");
        rwords.put("Class","Class");
        rwords.put("ClassLoader","ClassLoader");
        rwords.put("Compiler","Compiler");
        rwords.put("Double","Double");
        rwords.put("Float","Float");
        rwords.put("InheritableThreadLocal","InheritableThreadLocal");
        rwords.put("Integer","Integer");
        rwords.put("Long","Long");
        rwords.put("Math","Math");
        rwords.put("Number","Number");
        rwords.put("Object","Object");
        rwords.put("Package","Package");
        rwords.put("Process","Process");
        rwords.put("Runtime","Runtime");
        rwords.put("RuntimePermission","RuntimePermission");
        rwords.put("SecurityManager","SecurityManager");
        rwords.put("Short","Short");
        rwords.put("StackTraceElement","StackTraceElement");
        rwords.put("StrictMath","StrictMath");
        rwords.put("String","String");
        rwords.put("StringBuffer","StringBuffer");
        rwords.put("System","System");
        rwords.put("Thread","Thread");
        rwords.put("ThreadGroup","ThreadGroup");
        rwords.put("ThreadLocal","ThreadLocal");
        rwords.put("Throwable","Throwable");
        rwords.put("Void","Void");
        //whew, that was a lot

        /** finished building table */

    }


}

/**
 * Makes two passes:
 *
 * First pass: builds global symbol table.
 * -Ensure that no duplicate variable or function definitions are made.
 *
 * Second pass: performs checking on the following:
 * -repeated variable definitions
 * -ensuring that unknown identifiers are not referenced
 * -function arguement count match
 * -function return type
 * -a few other little picky things, including missing return statements to functions
```

```
 * -matrix invalid size declarators
 * -start method (no return statement allowed)
 * -built-in checking for all incorrectly used builtin functions
 * -checks that the return statement exists if function is not "void" (missing return
statement)
 * -assignment checking
 * -expression checking
 * -logical expression checking
 *
 * dead code checker: these cannot be detected in this version:
 *           include a cautionary note in the documentation
 *
 * not all errors can be detected.  the java compiler detects for 'possible'
 * undefined variables and sections of 'dead code'.  Because of the code generated
 * these messages are intelligable, but they are not caught here.
 *
 * also checked
 *
 */
buildGlobalTable // does not return anything
{
    //build new symbol tables
    globalTable = new BroomSymbolTable();
    context = new Context(globalTable);

    //fill the reserved word table with the reserved words
    buildrtable();
}
    : #( program:ID {cid(program);}  //note: cid checks if identifier is java keyword

            {globalScope=true;}   //globalScope tells rule 'variabledec' to use global
stack
            (variabledec)*
            { globalScope=false; }

            (EQUALS)* (funcdef)* START ) // note: will throw odd error if start does not
exist
        {
            if ( debugMode ) { //print global table contents if in debug mode.
                System.out.println("*** PASS ONE COMPLETE ***");
                System.out.println("GLOBAL TABLE CONTENTS AT END OF PHASE 1:");
                System.out.println(globalTable);
                System.out.println("\n\n");
            }
        };

variabledec
    : #(dec:DECLARATION ("Matrix"|"Number") (test:ID {cid(test);} )+ )
        {

            //System.out.println("Declaration with:" + dec.getNumberOfChildren() + "
children.");
            int numChildren = dec.getNumberOfChildren();

            AST typeSpecifier = dec.getFirstChild();
            String type = typeSpecifier.getText();
            //System.out.println("The type in this specific is: " +
typeSpecifier.getText());

            AST var = typeSpecifier.getNextSibling();
            for ( int i=1; i<numChildren; i++ ) {

                Symbol newSym = new Symbol(var.getText(),type);

                //System.out.println("Current symbol: "+ i+ "=" + newSym);

                /** Check for static error: duplicate variable declaration. */
                if ( context.in(newSym) ) {
                    //System.out.println("Semantic error on line :" );
                    err("variable " + newSym.getName() + " has already been defined.\n"
                        + "\t Please remove duplicate declaration.");
```

```
                } else {
                    if ( globalScope )
                        globalTable.add( newSym );   //add variables to global or local
scope
                    else {
                        context.addToLocal( newSym ); //depending on location of
declaration
                    }
                }

                var = var.getNextSibling();
            }

        };

funcdef
{
    context = new Context( globalTable );
}

    : #(f:FUNCTIONDEFINITION rtype:("Matrix"|"Number"|"Void")

            fname:ID { cid(fname); curFunc = fname.getText(); }

            ARGLIST
            (variabledec)* //local variable declarations
            STMTLIST
        )
        {

            //the following is a complex set of commands that really
            //just constucts a symbol out of the function and adds it to the symbol
            //table

            AST typeNode = f.getFirstChild();
            AST nameNode = typeNode.getNextSibling(); // the return type

            AST arglistNode = nameNode.getNextSibling();
            int numArgs = arglistNode.getNumberOfChildren();
            Symbol[] args = new Symbol[numArgs];

            if ( numArgs > 0 ) {

                AST argList = arglistNode.getFirstChild();
                for (int i=0 ;i<numArgs; i++) {
                    args[i] = new Symbol(
argList.getFirstChild().getNextSibling().getText(), //name
                                argList.getFirstChild().getText() ); //type
                    argList = argList.getNextSibling();
                }
            }

            Symbol newSym = new Symbol(nameNode.getText(), typeNode.getText(), args);

            /** Check for static error: function declaration. */
            if ( globalTable.in( newSym ) ) {
                err( "function " + newSym.getName() + " has already been defined.\n" +
                    "\t Please remove duplicate function declaration.");
                //System.out.println("\nERROR: attempted duplicate function definition: "
+ newSym.getName() );
                //System.out.println("Symbol: " + globalTable.get(fname.getText()) + "
already in table.");
            } else {
                    globalTable.add( newSym );
            }

        };

// SECOND PASS FUNCTION // SECOND PASS FUNCTION
// This checks for the rest of the errors (most of them)
```

```
// SECOND PASS FUNCTION // SECOND PASS FUNCTION

semanticCheck            //ignore declarations this time
    : #(program:ID {cid(program);} (DECLARATION)* (assignment)* (function)* startfunction
) //add support for START later
        {
            if ( debugMode ) {
                System.out.println("static semantic check complete."); }
        }
    ;

assignment
{
    int dest, src;
}
    : #(expr:EQUALS dest=expr src=expr )
        {
            //System.out.println("Found an assignment with " + expr.getNumberOfChildren()
+ " kids.");
            //System.out.println("Left is type: " + dest + " and right: " + src);

            /** Static check - make sure that the assignment types match */
            if (dest==src) {
                if (debugMode) {System.out.println("Assignment okay.");  }
            }
            else if (dest==Symbol.VOID) {
                err("left side of assignment is not a Number or Matrix.");
                //System.out.println("ERROR: left side of assignmnet invalid.");
            }
            else if (src==Symbol.VOID) {
                err("right side of an assignment is invalid.\n\t" +
                        "No value detected.  Make sure functions return value and are
operational.");
                //System.out.println("ERROR: right side of assignment is invalid.");
            }
            else {
                    //check for type mismatch on assignment
                String leftType = Symbol.getTypeN( dest );
                String rightType = Symbol.getTypeN( src );
                err("assignment type mismatch.\n\t " + leftType + " := " + rightType + "
is not valid.");
                //System.out.println("ERROR: assignment type mismatch detected: " + dest
+ "," + src);
            }
       }
    ;

startfunction
{
    context = new Context( globalTable );
    expectedReturnType = Symbol.VOID; //start method returns nothing
    curFunc = "start()";
    boolean r;
}
    : #(START (variabledec)* r=stmtlist)
    ;


function
{
    context = new Context( globalTable );
    boolean freturns = false;             //means that control flow must return
statements
}

    : #(f:FUNCTIONDEFINITION  ("Matrix"|"Number"|"Void")
            {
                AST rtype = f.getFirstChild();
                String returnType = rtype.getText();
                Symbol s = new Symbol("dummy", 0);
```

```
                expectedReturnType = s.findType( returnType );
            }

            fname:ID  { cid(fname);
                if (debugMode) {
                    System.out.println("*** Scanning function arguments for: " +
fname.getText() + ". ***");
                }
                curFunc = fname.getText();
                }

            (arglist)
            (variabledec)*
            freturns = stmtlist
            { if ( !freturns && expectedReturnType != Symbol.VOID )
                {
                    err("function " + fname.getText() + " is missing a return
statement.\n\t" +
                        "It expected to return " + Symbol.getTypeN(expectedReturnType) +
".");
                    //System.out.println("ERROR: function: " + fname.getText() + " is
missing a return statement.");
                }
            }

        )
        {
            if (debugMode) {
                System.out.println("Function " + fname.getText() + " finished check.");
                //System.out.println("Local context contents: " + context);
                System.out.println();
            }
        }
    ;

// add the local arguements passed to the function to the local context
arglist
    : #(ARGLIST (arg:ARGDECL
                {

                    AST typeNode = arg.getFirstChild();
                    AST nameNode = typeNode.getNextSibling();

                    String type = typeNode.getText();
                    //System.out.println("DEBUG: type specifier for function param: " +
type);

                    String name = nameNode.getText();
                    //System.out.println("DEBUG: the respective name for the function:" +
name);

                    /** Add static semantic checking here, besides internal
                        making sure that it has not been defined locally.  */
                    Symbol newSym  =new Symbol(name,type);

                    /** Static check NO DUPLICATES IN FUNCTION PARAMETER NAMES */
                    if ( context.in(newSym) ) {
                        err("function parameter " + newSym + " conflicts with global
variable " + context.get(name)
                            + "\n\t"
                            + "Please rename the function parameter.");
                    }
                    else {
                        context.addToLocal( new Symbol(name,type) );
                    }

                }
                )*
       )
    ;
```

```
stmtlist returns [boolean returnsValue]  //checks for missing return statement
{
    int type;
    returnsValue = false;
}
    : #(a:STMTLIST  { if ( debugMode ) System.out.println("Beginning of statement list");
}

            (
                b:"return"
                {
                    type = expr ( b.getFirstChild() );

                    if ( expectedReturnType == Symbol.VOID ) {
                        err("return statement in function of 'Void' return type.\n\t"
                            + " Please remove the return statement or give function
return type.");

                        //System.out.println("ERROR: return statement used in function of
void return type.");
                        //System.out.println("\t " + b.toStringList() + " should be
removed.");

                    } else if ( type != expectedReturnType ) {
                        err("mismatched return type.\n\t Returns type " +
Symbol.getTypeN( type )
                            + " but should expected to return type " + Symbol.getTypeN(
expectedReturnType ) );
                        //System.out.print("\nERROR: Mismatched return type");
                        //System.out.println("\t  " + b.toStringList() );
                    } else {
                        if ( debugMode ) {
                            System.out.println("return statement is okay.");
                        }
                    }

                    returnsValue = true;
                }

            | assignment

            | type=funcall

            | returnsValue=control_flow

            | PRINTTEXT  //no checking needed : handled by the grammar

            | #(PRINTVARIABLEVALUE varname:ID {cid(varname);} PRINT) {
                    Symbol s = context.get(varname.getText());
                    /** check to make sure that varname defined */
                    if ( s==null ) {
                        err("trying to print undefined variable " + varname.getText() );
                    }
                    //System.out.println("ERROR: " + varname.getText()
                    //  + "(that you are trying to print) is an undefined variable.");
                }
            )+

      )
    ;
expr returns [int type]
{
    int a,b;
    type=Symbol.VOID;

    int[] mval;

}
    : #(ADD a=expr b=expr )
        {
            if (a==b) type=a; else err("mismatched type on '+'");
```

```
        }
    | #(SUB a=expr b=expr )
        { if (a==b) type=a; else err("mismatched type on '-'"); }
    | #(MUL a=expr b=expr )
        {
                    if (a==b) {
                        type=a;
            } else if ( (a==Symbol.MATRIX && b==Symbol.NUMBER ) ||
                        (a==Symbol.NUMBER && b==Symbol.MATRIX )     ) {
                type = Symbol.MATRIX;
            }
            else {
                        err("mismatched type on '*'");
                    }
                }


    | #(DIV a=expr b=expr ) //make sure that these are not MATRICES! (can't divide
matrices)
        {
            if (a==Symbol.NUMBER && b==Symbol.NUMBER) { type=Symbol.NUMBER; }
            else {
                err("division attempted on non-scalar.\n\t This operation is not
permitted.");
            }
        }

    | (a=funcall)
        { type=a; }

    | a=builtin_funcall { type=a; }

    | varname:ID
        {
            cid(varname);
            Symbol s = context.get(varname.getText());
            if ( s==null ) {
                err("reference to undefined variable " + varname.getText());
            } else {
                int t = s.getType(); // return type
                //System.out.println("expr matcher: VARIABLE matched: " + s);
                type = t;
            }
        }

    | n:NUMBER
            { type=Symbol.NUMBER;  }

    | m:(mval=matrixcheck)
        { type=Symbol.MATRIX;  }

    ;

funcall returns [int type]    //check for missing return types
{
    type = Symbol.VOID;

    int[] paramtypes = new int[0]; //for testing function parameter matchings
}

    : #(FUNCALL f_id:ID {cid(f_id);} (paramtypes=expr_list)? )
        {
            String fname = f_id.getText();

            Symbol f = context.get( fname ); // f is the function called

            /** Static check - make sure that the function is defined */
            if ( !context.in( fname ) ) {
                err("reference to undefined function " + fname);
            } else {
```

```
                Symbol[] functionSymbols = f.getArgs();

                /** Static check - make sure that the number of arguments is correct. */
                if ( functionSymbols.length != paramtypes.length ) {
                    err("function call with incorrect number of arguments.\n\t" +
                        fname + " expected " + functionSymbols.length + ", but got " +
paramtypes.length + ".");
                }
                else {
                    /** Static check - make sure that argument types match definition of
function in context */
                    boolean pmismatch = false; int i;
                    for (i=0;i<paramtypes.length;i++) {
                        if ( paramtypes[i] != functionSymbols[i].getType() ) {
                            pmismatch = true;
                            break;
                        }
                    }
                    if ( pmismatch ) {
                        err("function call on " + f.getName() + " with type mismatch on
argument number " + i + ".\n");
                    } else {
                        if (debugMode) System.out.println("Function call " + f.getName()
+ " okay.");
                    } //end arg type

                } //end arg num check

                //pass back the return type of the function in any case except if the
function
                //is not defined.
                type = f.getType();

            } //end not defined check

        }
    ;


expr_list returns [int[] paramtypes] //returns the parameters as symbols so that they can
be tested
{
    int n,r;
    int i=0;
    paramtypes = new int[0];
}
    : #(p:EXPRLIST {n=p.getNumberOfChildren(); paramtypes=new int[n];}
                ( r=expr { paramtypes[i]=r; i++;  } )*  )
    ;


matrixcheck returns [int[] dimension]
{
    dimension = new int[2]; // m x n
    int m = 0;
    int n = 0; //number of columns
}
    : #(mroot:MATRIX (
                row:MATRIXROW
                {
                    int elem_count = row.getNumberOfChildren();
                    if (n==0) { n=elem_count; }
                    else if ( n!=0 && n!=elem_count ) {
                        err("invalid matrix specification.\n\t Expected a row with "
                            + n + " elements, but got " + elem_count);
                    }
                    m++;
                }

                )*
            {
```

```
                dimension[0] = m;
                dimension[1] = n; //set the dimensions that it returns
            }
        )
        ;


control_flow returns [ boolean returnsValue ]
{
    boolean t=false;
    returnsValue = false;
}
    : #( IFSTMT  conditional

            t=stmtlist
            { if (t) returnsValue = true; }

            (t=stmtlist { if (t) returnsValue = true; })?

        )  //else clause optional

    | #( WHILESTMT  conditional (t=stmtlist) { if (t) returnsValue = true; } )

    | #( FORSTMT  assignment conditional assignment (t=stmtlist) { if (t) returnsValue =
true; } )

    ;


conditional
    : #(NOT conditional )

    | #(AND conditional conditional  )

    | #(OR conditional conditional )

    | #(LOGICALEXPR logicalExpr)
    ;

logicalExpr
{
    int a,b;
}
    : #(lexpr:LOGGT  a=expr b=expr  )
        {
            if (a!=b) err("mismatched types on '>' logical operator.\n");
            /** > OP cannot be applied to matrices */
            if ( a==Symbol.MATRIX || b==Symbol.MATRIX ) {
                err("'>' operator cannot be applied to matrices.");
            }
        }
    | #(lexpr2:LOGLT  a=expr b=expr  )
        { if (a!=b) err("mismatched types on '<' logical expression.\n");
            /** < OP cannot be applied to matrices */
            if ( a==Symbol.MATRIX || b==Symbol.MATRIX ) {
                err("'<' operator cannot be applied to matrices.\n");
            }
        }
    | #(lexpr3:LOGEQ  a=expr b=expr  )
        { if (a!=b) err("mismatched types on '==' logical expression.\n"); }
    | #(lexpr4:LOGNEQ a=expr b=expr  )
        { if (a!=b) err("mismatched types on '!=' logical expression.\n"); }
    ;

builtin_funcall returns [int type]
{
    int[] dim;
    int p1, p2, p3, p4;
    AST arg, arg2, arg3, arg4;

    //  int[] p; //return types on the parameters
```

```
    //int i=0;
    type = Symbol.VOID;

    int t;
    /** Begin static semantic checking for builtin functions */
}
    : #(fcn:BUILTIN_FUNCALL (NOARYOP|UNARYOP|BINARYOP|TERNARYOP) ( t=expr )+ )
        {
            AST bf = fcn.getFirstChild();

            /** first, determine the basic category of builtin funcall */
            int c = bf.getType();

            int nargs = fcn.getNumberOfChildren() - 1;
            arg = bf.getNextSibling();

            AST callNode = bf.getFirstChild();
            int ftype = callNode.getType();
            String fname = callNode.getText();

            /** perform a semantic check on each **/

            // note: don't have to worry about the number of arguments, grammar checks
for that
            // checking included so that in case grammar changes, will work later

            switch (c) {

            case NOARYOP:
                if ( nargs != 1 )
                    err(fname + " expected 1 arg, but got " + nargs);

                /** coincidentally, all noaryop functions require matrices, so we check
here */
                p1 = expr ( arg );

                if ( p1 != Symbol.MATRIX )
                    err(fname + " needs a matrix as input.");

                switch (ftype) {
                case INV:
                    /** do the following check if the matrix is defined explicity in the
function */
                    if ( arg.getType() == Symbol.MATRIX ) {
                        dim = matrixcheck(arg);
                        //System.out.println("Dimensions detected: " + dim[0] + " x " +
dim[1]);
                        if (!isSquare(dim)) {
                            err("inv needs to be called on a square matrix.");
                        }
                    }
                    type = Symbol.MATRIX;
                    break;
                case TRANS:
                    type = Symbol.MATRIX;
                    break;
                case DET:
                case RANK:
                case NUMROWS:
                case NUMCOLS:
                    type = Symbol.NUMBER;
                }

                break;

            case UNARYOP:

                if ( nargs != 2 )
                    System.out.println(fname + " expected 2 args, but got " + nargs);

                arg2 = arg.getNextSibling();
```

```
                p1 = expr( arg );
                p2 = expr( arg2 );

                switch (ftype) {
                    /** Make sure the the inputs are both MATRICES */
                    case GAUSS:

                    if (p1!=Symbol.MATRIX) {
                        err("gauss(MATRIX, MATRIX) needs matrix for its input first
parameter.");
                    }
                    if (p2!=Symbol.MATRIX) {
                        err("gauss(MATRIX, MATRIX) needs matrix for its input second
parameter.");
                    }

                    type=Symbol.MATRIX;

                    /** Special type check: make sure input is square */
                    if (arg.getType() == MATRIX) {
                        dim = matrixcheck(arg);
                        if (debugMode) {
                            System.out.println("DEBUG Dimensions detected: " + dim[0] + "
x " + dim[1]);
                        }
                        if (!isSquare(dim)) {
                            err("gauss(A,b) needs to be called on a SQUARE matrix A.");
                        }

                        /** Now that we know dim of A, m x n, check the dimensions of the
b_col */
                        if ( arg2.getType() == MATRIX ) {
                            int[] b_col_dim = matrixcheck(arg2);

                            // make sure the A.m = b_col.m
                            if ( b_col_dim[0] != dim[0] ) {
                                err("for gauss(A,b) A.m != b.m.  A.m should be b.m");
                            }

                            // make sure that b_col.n = 1
                            if ( b_col_dim[1] != 1 ) {
                                err("for gauss(A,b) b.n != 1");
                            }
                        }
                    }
                    break;
                case ROW:

                    if ( p1 != Symbol.MATRIX ) {
                        err("row requires MATRIX as first argument.");
                    }
                    if ( p2 != Symbol.NUMBER ) {
                        err("row requires NUMBER as second argument.");
                    }

                    if ( arg.getType() == MATRIX && arg2.getType() == NUMBER ) {
                        dim = matrixcheck(arg);
                        int m = dim[0];
                        int i = Math.round( Float.parseFloat( arg2.getText() ) );
                        if ( (i<0) || (i>m) ) {
                            err("the row specified is out of bounds.");
                        }
                    }

                    type = Symbol.MATRIX;
                    break;
                case COL:

                    if ( p1 != Symbol.MATRIX ) {
                        err("col requires MATRIX as first argument.");
```

```
            }
            if ( p2 != Symbol.NUMBER ) {
                err("col requires NUMBER as second argument.");
            }

            if ( arg.getType() == MATRIX && arg2.getType() == NUMBER ) {
                dim = matrixcheck(arg);
                int n = dim[1];
                int i = Math.round( Float.parseFloat( arg2.getText() ) );
                if ( (i<0) || (i>n) ) {
                    err("The col specified is out of bounds.");
                }
            }

            type = Symbol.MATRIX;
            break;

            case ADDROW:  //todo: confirm that this is correct
            case ADDCOL:
            if ( p1 != Symbol.MATRIX ) {
                err(fname + " requires MATRIX as first argument.");
            }
            if ( p2 != Symbol.MATRIX ) {
                err(fname + " requires MATRIX as second argument.");
            }
            type = Symbol.MATRIX;
            break;
        case DELETEROW:
        case DELETECOL:
            if ( p1 != Symbol.MATRIX ) {
                err(fname + " requires MATRIX as first argument.");
            }
            if ( p2 != Symbol.NUMBER ) {
                err(fname + " col requires NUMBER as second argument.");
            }
            type = Symbol.MATRIX;
            break;

        case APPENDLR:
        case APPENDUD:

            if ( p1 != Symbol.MATRIX ) {
                err(fname + " requires MATRIX as first argument.");
            }
            if ( p2 != Symbol.MATRIX ) {
                err(fname + " requires MATRIX as second argument.");
            }
            type = Symbol.MATRIX;
            break;


        case NEWMATRIX:

            if ( p1 != Symbol.NUMBER ) {
                err(fname + " requires NUMBER as first argument.");
            }
            if ( p2 != Symbol.NUMBER ) {
                err(fname + " requires NUMBER as second argument.");
            }

            type= Symbol.MATRIX;

            break;

        }


        break;

    case BINARYOP:
        if ( nargs != 3 )
```

```
                    err("" + fname + " expected 3 args, but got " + nargs);

                arg2 = arg.getNextSibling();
                arg3 = arg2.getNextSibling();

                p1 = expr( arg );
                p2 = expr( arg2 );
                p3 = expr( arg3 );

                switch (ftype) {

                case ELEMENT:   //NUMBER <- ( MATRIX NUMBER NUMBER ) //check range of
numbers
                case ROWSWAP:   //MATRIX <- ( MATRIX NUMBER NUMBER ) //check range of
numbers
                case COLSWAP:   //MATRIX <- ( MATRIX NUMBER NUMBER ) //check range of
numbers
                    if ( p1 != Symbol.MATRIX ) {
                        err("on param 1, " + fname + " requires MATRIX, NUMBER, NUMBER as
arguments.");
                    }
                    if ( p2 != Symbol.NUMBER ) {
                        err("on param 2, " + fname + " requires MATRIX, NUMBER, NUMBER as
arguments.");
                    }
                    if ( p3 != Symbol.NUMBER ) {
                        err("on param 3, " + fname + " requires MATRIX, NUMBER, NUMBER as
arguments.");
                    }

                    if ( ftype == ELEMENT ) type = Symbol.NUMBER;
                    else type = Symbol.MATRIX;

                    break;

                case SETROW:    //MATRIX <- ( MATRIX MATRIX NUMBER ) //check range of
NUMBER, and that arg2 is column
                case SETCOL:    //MATRIX <- ( MATRIX MATRIX NUMBER ) //check input of

                    if ( p1 != Symbol.MATRIX ) {
                        err("on param 1, " + fname + " requires MATRIX, MATRIX, NUMBER as
arguments.");
                    }
                    if ( p2 != Symbol.MATRIX ) {
                        err("on param 2, " + fname + " requires MATRIX, MATRIX, NUMBER as
arguments.");
                    }
                    if ( p3 != Symbol.NUMBER ) {
                        err("on param 3, " + fname + " requires MATRIX, MATRIX, NUMBER as
arguments.");
                    }

                    type = Symbol.MATRIX;

                    break;

                }
                break;

            case TERNARYOP:

                arg2 = arg.getNextSibling();
                arg3 = arg2.getNextSibling();
                arg4 = arg3.getNextSibling();

                p1 = expr( arg );
                p2 = expr( arg2 );
                p3 = expr( arg3 );
                p4 = expr( arg4 );

                if ( nargs != 4 )
```

```
                    err("" + fname + " expected 4 args, but got " + nargs);

                //must be setelement, it is the only ternary operations
                //case SETELEMENT:  // MATRIX : ( MATRIX NUMBER NUMBER NUMBER )

                if ( p1 != Symbol.MATRIX ) {
                    err("on param 1, " + fname + " requires MATRIX, NUMBER, NUMBER,
NUMBER as arguments.");
                }
                if ( p2 != Symbol.NUMBER ) {
                    err("on param 2, " + fname + " requires MATRIX, NUMBER, NUMBER,
NUMBER as arguments.");
                    }
                if ( p3 != Symbol.NUMBER ) {
                    err("on param 3, " + fname + " requires MATRIX, NUMBER, NUMBER
,NUMBER as arguments.");
                }
                if ( p4 != Symbol.NUMBER ) {
                    err("on param 4, " + fname + " requires MATRIX, NUMBER, NUMBEr,
NUMBEr as arguments.");
                }

                type = Symbol.MATRIX;
                break;
            }
        }
    ;

// checks to make sure that the ID is not one of the java reserved
// words and also that it is not one of the basic java word languages
cid
    : j:ID
        {
            if ( rwords.containsKey( j.getText() ) ) {
                err("The word " + j + " is a reserved java keyword.");
            }
        }
    ;


revision 1.25
date: 2003/05/12 18:54:03;  author: mjw133;  state: Exp;  lines: +58 -14
Updated documentation
----------------------------
revision 1.24
date: 2003/05/09 19:45:29;  author: mjw133;  state: Exp;  lines: +1 -1
Removed an unnecessary print statement.
----------------------------
revision 1.23
date: 2003/05/05 01:30:58;  author: mjw133;  state: Exp;  lines: +18 -2
I don't know what I changed.
----------------------------
revision 1.22
date: 2003/05/04 20:07:59;  author: mjw133;  state: Exp;  lines: +2 -1
Fixed a smaller picky formatting error.
----------------------------
revision 1.21
date: 2003/05/04 20:07:13;  author: mjw133;  state: Exp;  lines: +2 -0
Now it should work.
----------------------------
revision 1.20
date: 2003/05/04 20:06:45;  author: mjw133;  state: Exp;  lines: +178 -119
Added vector list of errors and improved messages that
give you the function name of the error.  This version
is not yet functional.  I have broken something. will fix.
----------------------------
revision 1.19
date: 2003/05/04 15:06:27;  author: mjw133;  state: Exp;  lines: +6 -2
Fixed an error that was introduced by the previous edit.
The current revision should no longer be confused by
many local variable declarations on the first pass.
```

```
----------------------------
revision 1.18
date: 2003/05/04 15:02:04;  author: mjw133;  state: Exp;  lines: +2 -1
Removed local variable declaration checking from
the primary phase of the static semantic checker.
----------------------------
revision 1.17
date: 2003/05/03 20:47:16;  author: mjw133;  state: Exp;  lines: +20 -10
Fixed a few bugs related to null argument lists, void return types,
an error in the builtin_function calls.
----------------------------
revision 1.16
date: 2003/05/03 15:52:35;  author: mjw133;  state: Exp;  lines: +1 -0
Correct a missing '}' that was preventing the program from compiling.
----------------------------
revision 1.15
date: 2003/05/03 15:51:22;  author: mjw133;  state: Exp;  lines: +29 -15
Added support for the built-in print statements.
----------------------------
revision 1.14
date: 2003/05/03 04:00:44;  author: mjw133;  state: Exp;  lines: +122 -7
Got the ID checking for java reserved words to work.
Also added the matrix type checking functionality.
Everything should be ready except for line numbers of errors
and print statement scanning.
----------------------------
revision 1.13
date: 2003/05/03 03:11:29;  author: mjw133;  state: Exp;  lines: +15 -131
This version compiles, but the individual semantic
checking for all of the nodes has not been completely checked yet.
This needs to be done.
----------------------------
revision 1.12
date: 2003/05/03 03:04:08;  author: mjw133;  state: Exp;  lines: +304 -22
Not quite working yet...lot's of code to insert for the stupid issues of the builtin
functions.
----------------------------
revision 1.11
date: 2003/05/02 23:48:51;  author: mjw133;  state: Exp;  lines: +72 -18
Still working on the adding the builtin functions.
----------------------------
revision 1.10
date: 2003/05/02 20:25:54;  author: mjw133;  state: Exp;  lines: +85 -14
ADding builtin functions, but not finished yet.
----------------------------
revision 1.9
date: 2003/04/28 21:16:04;  author: mjw133;  state: Exp;  lines: +39 -19
Control flow added.
Static checker in semi-functional stage at this point.
The error messages need to be improved so that the text
from the error is reported.
Also, in this version, the print statements are not yet
supported.
----------------------------
revision 1.8
date: 2003/04/28 17:58:17;  author: mjw133;  state: Exp;  lines: +27 -12
--> Added support for checking that a return statement must exist
somewhere in a function if it is not void.
--> Added, if a function is void, and uses a return statement, print error msg
----------------------------
revision 1.7
date: 2003/04/28 17:30:27;  author: mjw133;  state: Exp;  lines: +22 -3
Added checking of matrix row sizes.  Tested on one example:
2pass.3expr.txt and appears to be given proper error messages and clean output.
----------------------------
revision 1.6
date: 2003/04/28 03:29:21;  author: mjw133;  state: Exp;  lines: +30 -8
Tested the previous revision and worked out a few inconsistencies,
the features stated in the previous revision or 2 should be
working.
----------------------------
```

```
revision 1.5
date: 2003/04/28 02:40:44;  author: mjw133;  state: Exp;  lines: +171 -52
Fixed bug that was causing a null pointer exception with calling undefined functions.
Improved the error messages to remove anything inappropriate.
This version has undergone some testing, but not thorough testing.
----------------------------
revision 1.4
date: 2003/04/27 18:34:05;  author: mjw133;  state: Exp;  lines: +117 -130
Updated so that the following are now working:
expr type checking with */-+ operators
assignment type checking
no duplicate
variable not defined

the usage is limited.  in this version, statementlists are not allowed.
----------------------------
revision 1.3
date: 2003/04/26 21:24:57;  author: mjw133;  state: Exp;  lines: +117 -23
Not finished.
----------------------------
revision 1.2
date: 2003/04/26 17:31:52;  author: mjw133;  state: Exp;  lines: +159 -7
Added handling for expressions.  Some of the work is done, but more remains to be done.
----------------------------
revision 1.1
date: 2003/04/25 20:47:59;  author: mjw133;  state: Exp;
Initial revision
=============================================================================
```

The following classes are needed by the treewalker: (also by Michael Weiss)

```
/**
 * Represents a symbol in the symbol table
 * These symbols correspond to Identifiers in BROOM
 *
 * @author Michael Weiss
 */
public class Symbol
{
    public Symbol( String name, int type ) {
        this.name = name;
        this.type = type;
        isFunction = false;
    }

    public Symbol( String name, String type ) {
        this.name = name;
        this.type = findType(type);
        isFunction = false;
    }

    /**
     * Special constructor for a function symbol.
     * @type is the return type of the function
     * @param args are the function arguments, presumably not functions
     */
    public Symbol( String name, int type, Symbol[] args) {
        isFunction = true;
        this.type = type;
        this.name = name;
        this.args = args;
    }

    public Symbol( String name, String type, Symbol[] args) {
        isFunction = true;
        this.type = findType(type);
        this.name = name;
        this.args = args;
    }
```

```
public String getName() {
    return name;
}

public int getType() {
    return type;
}

/**
 * Return true if the symbol is a function
 */
public boolean isFunction() {
    return isFunction;
}

/**
 * Returns the numeric value corresponding to the type t
 */
public int findType(String type) {
    int x;
    if ( type.equalsIgnoreCase("Matrix") ) {
        x = MATRIX;
    } else if (type.equalsIgnoreCase("Number")) {
        x = NUMBER;
    } else if (type.equalsIgnoreCase("void")) {
        x = VOID;
    } else {
        System.out.println("ERROR: unknown type specification." + type);
        x = 0;
    }
    return x;
}

/**
 * Reverse lookup to findType
 */
public String getTypeName (int type) {
    switch (type) {
    case VOID: return "void";
    case NUMBER: return "number";
    case MATRIX: return "matrix";
    }
    return "error, type not defined.";
}


/**
 * Reverse lookup to findType
 */
public static String getTypeN (int type) {
    switch (type) {
    case VOID: return "void";
    case NUMBER: return "number";
    case MATRIX: return "matrix";
    }
    return "error, type not defined.";
}


public String toString() {

    if (!isFunction) {
        return "Symbol: " + name + ":" + getTypeName(type);
    }
    else {
        String o = new String();
        o+="Symbol: Function " + name + " returns " + getTypeName(type)
            + " with args:\n";
        for (int i=0;i<args.length;i++) {
            o+=args[i].toString() + "\n";
        }
```

```
            return o;
        }
    }


    /**
     * Returns the array of arguemnt symbols
     */
    public Symbol[] getArgs() {
        return args;
    }

    public static final int VOID = 0;   //no type defined
    public static final int NUMBER = 1;
    public static final int MATRIX = 2;

    private String name;
    private int type;
    private boolean isFunction = false;

    //the following should not be null only if symbol represents function
    private Symbol[] args;
}

import java.util.*;

/**
 *
 * Wrapper class for hash table which is used as a SymbolTable
 *
 * @author Michael Weiss
 */
public class BroomSymbolTable
{
    /**
     * Constructor
     */
    public BroomSymbolTable() {
        st = new Hashtable();
    }

    /**
     * Add a symbol to the table
     */
    public void add( Symbol s ) {
        if ( in(s) ) {
            //TODO: maybe implement error as an exception thrown !!!WITH LINE NUMBER!!!
            System.out.println("Error: duplicate symbol entry attempted on:\n" + s);
        } else {
            st.put(s.getName(), s);
        }
    }

    /**
     * Object in the table?
     */
    public boolean in( Symbol s ) {
        return st.containsKey(s.getName());
    }

    /**
     * Object in table by name?
     */
    public boolean in( String sname ) {
        return st.containsKey(sname);
    }

    /**
     * Lookup directly the type of an object
     */
    public int typeOf( String sname ) {
```

```
        Symbol s = (Symbol) st.get(sname);
        return s.getType();
    }

    /**
     * Lookup the object itself
     */
    public Symbol get( String sname ) {
        return (Symbol) st.get(sname);
    }

    /**
     * Get the entire contents of the hashtable
     */
    public String toString() {
        String r = new String();

        r += "Begin table contents:\n";

        Collection c = st.values();
        Symbol[] allContents = (Symbol[]) c.toArray( new Symbol[c.size()] );

        for (int i=0;i<allContents.length;i++) {
            r+=String.valueOf(i)+ "-->" +allContents[i].toString()+ ".\n";
        }

        r += "End table contents.";

        return r;
    }

    private Hashtable st;
}


/**
 * A small wrapper class to both a global and a local
 * context.
 *
 * @author Michael Weiss
 */
public class Context
{
    public  BroomSymbolTable global;

    //direct access to local not permitted
    //why? don't want to accidentally add something to local
    //context that is also in the global context.  Control should
    //be automatic.
    private BroomSymbolTable local;

    public Context (BroomSymbolTable g ) {
        this.global = g;
        local = new BroomSymbolTable();
    }

    /**
     * Add the Symbol to the local context
     *
     * Checks to make sure that the object is not already defined
     * locally and that it is also not defined globally.
     *
     * If either of these conditionals occur, then it will not
     * add the symbol to the context, otherwise it will.
     */
    public void addToLocal( Symbol s ) {
        if ( global.in(s) ) {
            System.out.println( "Cannot add " + s + " to context." );
            System.out.println( "It is already in the globalTable.");
        }
        else if ( local.in(s) ) {
```

```
            System.out.println( "Cannot add " + s + " to context." );
            System.out.println( "It is already in the *local* context." );
        } else {
            //if here, everything is okay to add
            local.add( s );
        }
    }

    // in
    public boolean in( Symbol s) {
        return (global.in(s) || local.in(s));
    }

    public boolean in( String s) {
        Symbol test = get(s);
        if ( test == null ) { return false; }
        else { return true; }
    }

    // get
    public Symbol get( String s) {
        //first, look in the local context
        Symbol r = local.get(s);

        //if it's not there look in the global context
        if (r==null) {
            r = global.get(s);
        }

        return r;
    }

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

}
```

---

CODE GENERATION  - Chris Tobin

---

```
//4-26-03, cmt69, beginning code generator tree walker
//4-27-03, cmt69, continuing code generator
//5-01-03, cmt69, major problem hit, {1,2,3} array declarations can only
//                be used at initialization, so taking Edwards' suggestion
//                to have temp variables.  really changing all the rules.
//                If this doesnt work, check codegenerator1.g, this contains
//                the previous version of codegenerato grammar.
{import java.io.*;}
class CodeGenerator extends TreeParser;
options
{
    importVocab = broom;

}

{
    //this is the string that will be printed as the output file
    public String programString;
    //simple integer that contains the number of tabs that you are using
    int tabs;
    //integer that will be used to generate unique temporary variable names
    int unique =0;
    String programName;
}

//thus begins the code generation
```

```
buildCode
{
    programString ="//Thus begins your broom program \n";
    tabs =0;
}

    :#(
            i: ID
            {
                programName = i.getText();
                programString += "public class "+programName+" \n{\n";
            }

            //global variable declaration
            (variabledec)*

            //before the assignments you start up the constructor
            {
                programString += "public "+programName+"()\n{\n";
                tabs++;
            }

            //assignments
            (
                {String temp ="";}
                temp = equalsrule
                {programString += temp;}
            )*

            //close off the constructor
            {
                programString += "}\n";
                tabs--;
            }

            //BEGIN MATCHING THE FUNCTION DEFINITIONS IF THEY ARE THERE
            (functiondefinition)*

            startfunction
            //now you want to put it all together in the main method.
            //1) create new object,
            //2) call start function
            {
                programString += "public static void main(String[] args)\n{";
                programString += "\t"+programName+" myProgram = new
"+programName+"();\n";
                programString += "\t"+"myProgram.start();\n";
                programString += "}\n";

                programString += "}";
            }

        )
        //here is where you would want to print out to the file, right now just print out
the
        //the string to standard out
        //5-9-03, cmt69, stoppping printint to standard out
//        {System.out.println(programString);}

        //5-2-03, cmt69, now add in the ability to print out to the file that they have
specified
        {
            try{
                BufferedWriter out = new BufferedWriter( new FileWriter
(programName+".java"));
                out.write(programString);
                out.close();
            }
            catch(IOException e)
            {
                System.out.println(e);
```

```
            }
        }
    ;

variabledec

    :#( dec:DECLARATION ("Matrix"|"Number") (ID)+)
        {
            //holds the number of children, one for the type, the rest for the variables
            int numChildren = dec.getNumberOfChildren();
            //type holds the type of it
            AST typenode = dec.getFirstChild();
            String type = typenode.getText();

            //check to see what the type is and write the appropriate java type.
            if(type.equals("Matrix"))
                programString += "BroomMatrix ";
            else
                programString += "float ";


            for (int i = 1; i< numChildren; i++)
            {
                typenode = typenode.getNextSibling();
                programString += " "+typenode.getText();
                if(i == numChildren-1)
                {}
                else
                {
                    programString +=",";
                }
            }
            //add in the semi colon and youre all set
            programString += ";\n";
        }


    ;



/*------------------------------------------------------------------------
cmt69, 4-27-03, now beginning the code that is used for the main body of program,
not only the stuff for globals.
------------------------------------------------------------------------*/
//5-4-03, cmt69, changed so that program has to define void type.

functiondefinition
    {
        String copyString ="";
    }
:
    #(
            f : FUNCTIONDEFINITION
            ("Matrix"|"Number"|"Void")
            {

                tabs++;
                String t = "";
                for(int j = 0; j< tabs; j++)
                    t+="\t";

                programString += "public ";
                AST type = f.getFirstChild();
                String name = type.getText();
                if( name.equals("Matrix"))
                    programString += "BroomMatrix  ";
                else if  (name.equals("Number"))
                    programString += "float ";
                else
                    programString += "void ";
```

```
            }
            i : ID
            {
                programString += i.getText()+" (";

            }

            copyString = arglist

            {programString += ")\n{\n";}

            {programString += t+" "+copyString;}
            (
                {programString+= t;}
                variabledec
            )*

            stmtlist

            //finally add in the closing bracket
            {
                programString += "}\n\n";
                tabs--;
            }
     )
    ;



startfunction:
    #(
            f : START
            {

                tabs++;
                String t = "";
                for(int j = 0; j< tabs; j++)
                    t+="\t";

                programString += "public void start() \n{\n";
            }


            (
                {programString+= t;}
                variabledec
            )*

            stmtlist

            //finally add in the closing bracket
            {
                programString += "}\n\n";
                tabs--;
            }
     )
    ;



stmtlist: #(STMTLIST (stmt)+)
    ;

stmt
        {
            String stmtString="";
            String t = "";
                for(int j = 0; j< tabs; j++)
                    t+="\t";
```

```
        }
    :
        {String temp;}
        temp = equalsrule
        {programString += temp;}

        |#("return"
            {
                programString += t+"return ";
            }

            stmtString = expr
            {
                programString += stmtString+";\n";
            }
          )

        |stmtString = funcall
            {programString += stmtString+";\n";}

        //5-1-03, cmt69, adding in the ability to print text
        |#(PRINTTEXT texttowrite:TEXT

            {programString += t+"System.out.println("+texttowrite.getText()+");\n";}
        )

        |#(PRINTVARIABLEVALUE
            myid : ID
            PRINT

            {
                //this function prints out the variable's value
                //if its a matrix, it prints out the formatted matrix text.
                //if its a number, it prints the number followed by a space, with no
newline character.
                String name = myid.getText();
                programString += t+"BroomMatrix.printVariable("+name+");\n";
            }

        )

        | #(p :IFSTMT

            {String tempString ="";}
            tempString = conditional

            //going to start if code here to give the conditional
            //the ability to have temp variables declared outside
            //of the the ifstmt.
            {
                programString += t+"if( "+tempString+")\n";
                programString += t+"{\n";
                tabs++;
            }

            stmtlist

            {
                programString += t+"}\n";
                programString += t+"else\n"+t+"{\n";
            }


            (stmtlist)?
            {
                programString += t+"}\n";
                tabs--;
            }
```

```
           )

        |#(WHILESTMT
            {String tempString ="";}
            tempString = conditional
            //again printing the code after the conditional is evaluated so that it can
print
            //temp variables outside the loop before it starts
            {
                programString += t+"while( "+tempString+")\n";
                programString += t+"{\n";
                tabs++;
            }

            stmtlist

            {
                programString += t+"}\n";
                tabs--;
            }
        )

        |#(FORSTMT
            {String ass1,cond,ass2;}
            ass1 = equalsrule
            cond = conditional
            ass2 = equalsrule
            {
                //note, all the equalsrules end with semis and a newline, so use String
replace method
                //to get rid of these things
                ass1 = ass1.replace('\n',' ');
                ass1 = ass1.replace('\t',' ');
                ass2 = ass2.replace('\n',' ');
                ass2 = ass2.replace(';',' ');
                ass2 = ass2.replace('\t',' ');
                programString +=t+"for("+ass1+cond+"; "+ass2+")\n"+t+"{\n";
                tabs++;
            }
            stmtlist

            {
                programString += t+"}\n";
                tabs--;
            }
        )
    ;


conditional returns [String conditionalString]
      {
        conditionalString = "";
        String tempString = "";
      }

    : conditionalString = logicalExpr1

    | #(AND
            conditionalString = logicalExpr1
            tempString = logicalExpr1
            {
                conditionalString += " && "+tempString;
            }
        )
    | #(OR
            conditionalString = logicalExpr1
            tempString = logicalExpr1
            {
                conditionalString += " || "+tempString;
            }
        )
```

```
    ;

logicalExpr1 returns [String logicalString]
     {
        logicalString ="";
        String tempString = "";
     }
    :#(LOGICALEXPR logicalString = logicalExpr)

    |#(NOT  #(LOGICALEXPR tempString = logicalExpr
            {
                logicalString += "!( "+tempString+" )";
            }
            )
        )
    ;


logicalExpr returns [String logicalString]
        {
            logicalString ="";
            String tempString ="";
        }
    :            //match the logical operator tree
            (#(LOGGT
                tempString = expr
                {logicalString += tempString+" > ";}
                tempString = expr
                {logicalString += tempString;}
             ))
            | (#(LOGLT
                tempString = expr
                {logicalString += tempString+" < ";}
                tempString = expr
                {logicalString += tempString;}
               ))
            | #(LOGEQ
                tempString = expr
                {logicalString += "BroomMatrix.equals("+tempString+", " ;}
                tempString = expr
                {logicalString += tempString+")";}
               )
            | #(LOGNEQ
                tempString = expr
                {logicalString += "BroomMatrix.notEquals("+tempString+", ";}
                tempString = expr
                {logicalString += tempString+")";}
              )

    ;

arglist returns [String copyString]
           {
            copyString = "";
            String tempString;
           }
    :
        #(ARGLIST
            (
                tempString= argdecl
                {copyString += tempString;}
                (
                    {programString += ",";}

                    tempString = argdecl
                    {copyString += tempString;}
                )*
            )?
        )
    ;
```

```
argdecl returns [String argdeclString]
    {
        argdeclString = "";
    }
    :
        #(a:  ARGDECL ("Matrix"|"Number") ID)
        {
            AST type = a.getFirstChild();
            AST arg  = type.getNextSibling();
            //add the type text to the string
            String name = type.getText();
            String argName = arg.getText();
            if(name.equals("Matrix"))
            {
                programString += " BroomMatrix";
                argdeclString += " "+argName+" = BroomMatrix.copy("+argName+");\n";
            }
            else
            {
                programString += " float";
            }
            programString += " "+arg.getText();
        }
    ;


equalsrule returns [String equalsruleString]
        {
            equalsruleString = "";
            String exprString;
        }
    :
        #(EQUALS i:ID

               exprString = expr

            {
                String t = "";
                for(int il = 0; il< tabs; il++)
                t+="\t";

                equalsruleString +=t;

                equalsruleString += i.getText()+" = ";

                equalsruleString+= exprString +"; \n";
            }

        )

    ;


//5-01-03, cmt69, the rules are now being changed so that every expression returns
// a string which is what should be added to the program string.  the only
// things that are written directly to the program string in this rule are the
// temporary variables.
expr returns [String exprString]
         {
            //this string will contain the expr call
            exprString = "";
            String tempString = "";
         }

       //simple #
       //just return the string that represents the #
       : ( n:NUMBER)
            {
                if(n.getNumberOfChildren() == 0)
                    exprString += "(float) "+n.getText();
                else
```

```
            exprString += "(float) -"+n.getText();
    }
//simple id
//just return the ID string back
|( i : ID)
    {
        //you have an identifier, so add the text of the id's name
        exprString += i.getText();
    }


/*NOTE: cmt69, 4-27-03.  In the java library that we make, we will have
an overloaded function for the arithmetical operators, one that takes
2 matrices and one that takes 2 floats.  This way, there is no need
for a symbol table in the code generator.*/

//cmt69, 5-1-03, adding in the fact that these rules will return
//expr string which is what should be put into programString
//checking to see if it is addition
|#(ADD
    {exprString+="BroomMatrix.add(";}
   tempString = expr
    {exprString += tempString+",";}
   tempString = expr)
    {exprString += tempString+")";}

//checking to see if its subtraction
|#(SUB
    {exprString += "BroomMatrix.subtract(";}
   tempString = expr
    {exprString += tempString+",";}
   tempString = expr
    {exprString += tempString+")";}
)

//checking to see if you have multiplication
|#(MUL
    {exprString += "BroomMatrix.multiply(";}
   tempString = expr
    {exprString += tempString+",";}
   tempString = expr
    {exprString += tempString+")";}
    )
//checking to see if you have division,
//semantic checker is making sure that no arguments to division are matrices
//this means that we can just plug in the java /

|#(DIV
    //make sure to cast just in case
    {exprString += "(float)";}
   tempString = expr
    //simple divide operation followed by the float cast
    {exprString += tempString + "/ (float)";}
   tempString = expr
    {exprString += tempString;}
)

//5-1-03, cmt69, when youre matching the builtin funcalls,
// you want to return the string that has these calls in them
// not print out to the program string
//I dont need to worry about the number of args because that is taken
//care of by grammar and by static semantics
|#(functionname:BUILTIN_FUNCALL
    (NOARYOP|UNARYOP|BINARYOP|TERNARYOP)

    //now get the name of the function that youre calling
    //and add it to the  exprString
        {
            AST testnode = functionname.getFirstChild();
            testnode = testnode.getFirstChild();
            exprString +=" BroomMatrix."+testnode.getText()+"(";
```

```
            }
        //your first expression you just want plain text

        tempString = expr

        {
            exprString += tempString;
        }

        //any other expressions you match, you wanna plug in a comma as well
        (

            tempString = expr
                {
                    exprString += ", "+ tempString;
                }
        )*

        //finally, close up your parenthesis
        {exprString+=")";}
      )

    //HERE IS WHERE YOU ARE USING THE TEMP VARIABLES
    |#(matroot: MATRIX
        {
            //start with a call to the constructor, it will take the temporary 2-d
array
            exprString +=" new BroomMatrix( ";
        }

            //HERE IS WHERE YOU ARE DECLARING THE TEMPORARY 2-D ARRAY
            //FIRST YOU WANNA DECLARE, THEN ASSIGN
            //add the declaration to the PROGRAM STRING

            {                               //make the temp var name

                String t = "";
                for(int il = 0; il< tabs; il++)
                t+="\t";

                programString += t+"float[][] _temp"+unique+" = {";
            }

        tempString = matrixrow

            {
                programString += tempString;
            }

        (
            //{programString += ", ";}
            tempString = matrixrow
                {programString += ", "+tempString;}
        )*

            //add a semi colon to the temp declaration
            {
                programString += "};\n";
            }


        //add in temp name to exprString along with closing paren
        {
            exprString += "_temp"+unique+")";
            //increment unique so that its not used again
            unique++;
        }
    )


    //funcall returns a string now as well
```

```
        | exprString = funcall

    ;

funcall returns [String funcallString]
    {
       funcallString ="";
       String tempString = "";
    }

    :#(FUNCALL
            id: ID
                {
                    //print out the name of the function
                    funcallString += id.getText()+"(";
                }
            (tempString = exprlist)?
                {
                    funcallString += tempString+ ")";
                }
           )
    ;

exprlist returns [String exprlistString]
        {
            exprlistString = "";
            String tempString = "";
        }

    : #(EXPRLIST
                //no comma beforehand
                tempString = expr
            {exprlistString += tempString;}
                (
                    //{programString += ", ";}
                    tempString = expr
                    {exprlistString += ", "+tempString;}
                )*
            )
    ;


matrixrow returns [String matrixrowString]
        {
            matrixrowString = "";
            String tempString = "";
        }
:
        #(MATRIXROW
            //add in the beginning curly bracket
            {matrixrowString += "{";}
            //match the first expression and add in the commas afterwards
            tempString = expr
            {matrixrowString += tempString;}
            (
                //{programString += ", ";}
                tempString = expr
                {matrixrowString += ", "+tempString;}
            )*
            //add the closing curly bracket
            {matrixrowString+= "}";}
        )
;



First Version of Code Generator.  So far it looks good, though im sure bugs, needs
extensive testing.
----------------------------
revision 1.8
date: 2003/05/09 17:32:47;  author: cmt;  state: Exp;  lines: +2 -1
```

```
changed so that code isnt printed to standard out.
----------------------------
revision 1.7
date: 2003/05/05 02:16:12;  author: cmt;  state: Exp;  lines: +32 -8
changed so that every function definition copies all the args so that pass by value is
allowed
----------------------------
revision 1.6
date: 2003/05/04 23:21:36;  author: cmt;  state: Exp;  lines: +2 -1
simple change to tree walker so that programs now have void type
----------------------------
revision 1.5
date: 2003/05/02 19:28:37;  author: cmt;  state: Exp;  lines: +22 -5
Removed some bugs where 'Matrix' would be printed instead of BroomMatrix.
- now it prints out to file whose name is root of AST
- now it imports the class BroomMatrix.java.  this needs to be changed to a package
----------------------------
revision 1.4
date: 2003/05/02 17:30:53;  author: cmt;  state: Exp;  lines: +2 -0
tried to change the builtin funcs so that they would subtract one from the
any indexes so that they are zero indexed for gabes methods.  couldnt
do this cause i dont worry about number of args, since it works
ok without knowing bc grammar and ssc take care of it.
----------------------------
revision 1.3
date: 2003/05/02 16:17:44;  author: cmt;  state: Exp;  lines: +10 -2
Changed so that start in Broom program becomes a method called start.
The main method now calls the constructor, and then start() on the
constructed object.
----------------------------
revision 1.2
date: 2003/05/02 03:04:05;  author: cmt;  state: Exp;  lines: +8 -2
few errors where the generated code would print NUMBER instead of float, think its fixed
----------------------------
revision 1.1
date: 2003/05/02 02:50:19;  author: cmt;  state: Exp;
Initial revision
=============================================================================



-----------------------------------------------------------------------------
THE BACK END (BroomMatrix.java utility)  Gabriel Glaser
-----------------------------------------------------------------------------
import java.util.*;

/**
 * BroomMatrix.java
 * @author Gabe Glaser
 *
 */


public class BroomMatrix
{

    float matrix[][];
    int m,n;

    /**
     * BroomMatrix constructor
     *
     * @param a A two dimemsional array that will populate the matrix
     */

    public BroomMatrix(float a[][])
    {
        matrix=a;
        m=matrix.length;
        n=matrix[0].length;
```

```
    }


    /**
     * makes a new matrix, filled with zeros
     *
     * @param m1 height of new matrix
     * @param n1 width of new matrix
     */

    public static BroomMatrix newMatrix(float m1, float n1)

    {
        int m2=Math.round(m1);
        int n2=Math.round(n1);

        float temp[][]=new float[m2][n2];
        return new BroomMatrix(temp);

    }


    /**
     * duplicates a matrix and returns the copy
     *
     * @param a the matrix to be copied.
     */

    public static BroomMatrix copy(BroomMatrix a)
    {
        float[][] tempMat=new float[a.m][a.n];
        for (int i=0;i<a.m;i++)
            for (int j=0;j<a.n;j++)
                tempMat[i][j]=a.matrix[i][j];

        return new BroomMatrix(tempMat);

    }

    /**
     * returns the inverse of a matrix, using the invC method to calculate.
     *
     * @param a the matrix to be inverted.
     */


    public static BroomMatrix inv(BroomMatrix a)
    {
        BroomMatrix dup=BroomMatrix.copy(a);
        if (BroomMatrix.rank(a)==a.m && a.m==a.n)
            {
                BroomMatrix result=BroomMatrix.invC(dup);
                return result;
            }
        else
            {
                System.out.println("Matrix :");
                BroomMatrix.printVariable(a);
                System.out.println("is not invertible.  Invertibility requires a square,
full rank matrix");
                e("");

                return null;
            }

    }


    /**
     * calulates inverse of a matrix.
     *
```

```
     * @param a the matrix to be inverted.
     */

    public static BroomMatrix invC(BroomMatrix a)
    {
        BroomMatrix dup=BroomMatrix.copy(a);
        float mat[][]=dup.matrix;
        float inverse[][]=new float[dup.m][dup.n];
        for(int i=0;i<a.m;i++)
            for (int j=0;j<a.m;j++)
                if (i==j)
                    inverse[i][j]=1;
                else inverse[i][j]=0;
        BroomMatrix in=new BroomMatrix(inverse);
        float tempRow[]=new float[dup.n];

        for (int i=0;i<dup.n;i++)
            {

                for (int j=i+1;j<dup.m;j++) // swap rows
                    {
                        if (Math.abs(dup.matrix[i][i])<Math.abs(dup.matrix[j][i]))
                            {dup=BroomMatrix.rowSwap(dup,i,j);
                            in=BroomMatrix.rowSwap(in,i,j);
                            }

                    }//rows have been swapped

                float pivot=dup.matrix[i][i];

                if ((new Float(pivot)).isNaN()||(new Float(pivot)).isInfinite())
                    {
                        System.out.println("singular");
                        System.out.println(pivot);
                        return null;

                    }

                //for each row under current, get a multiplier and apply it
                for (int k=i+1;k<a.m;k++)
                    {
                        if(true)//dup.matrix[k][i]!=0)
                            {
                                float m=dup.matrix[k][i]/pivot;

                                for (int l=0;l<a.n;l++) //for each element in the row
                                    {
                                        dup.matrix[k][l]=dup.matrix[k][l]-
m*dup.matrix[i][l];
                                    }

                                for (int l=0;l<a.n;l++) //for each element in the row
                                    {
                                        in.matrix[k][l]=in.matrix[k][l]-m*in.matrix[i][l];
                                    }
                            }
                    }//now have an upper triangular matrix.


            }

        for (int i=a.m-1;i>=0;i--)
            {


                float pivot2=dup.matrix[i][i];//get column pivot
                if ((new Float(pivot2)).isNaN()||(new Float(pivot2)).isInfinite())
                    {
                        System.out.println("singular");
                        System.out.println(pivot2);
                        return null;
```

```
            }


        for (int j=i-1;j>=0;j--) //for each row above..
            {
                float m2=dup.matrix[j][i]/pivot2; //get multiplier for row

                for (int k=0;k<a.n;k++)
                    {
                        dup.matrix[j][k]=dup.matrix[j][k]-m2*dup.matrix[i][k];
                        in.matrix[j][k]=in.matrix[j][k]-m2*in.matrix[i][k];
                    }
            }


        }

    for (int i=0;i<a.m;i++)
        for (int j=0;j<a.n;j++)
            in.matrix[i][j]=in.matrix[i][j]/dup.matrix[i][i];

    return in;
}




/**
 * performs gaussian elimination (Ax=b) and returns "x"
 *
 * @param a "A" in Ax=b
 * @param b the "b" vector in "Ax=b"
 *
 */
public static BroomMatrix gauss(BroomMatrix a,BroomMatrix b)
{

    if (a.m==a.n && b.m==a.m && b.n==1 && BroomMatrix.rank(a)==a.m)
        {
            BroomMatrix dup=BroomMatrix.copy(a);
            //float mat[][]=dup.matrix;
            float inverse[][]=new float[dup.m][dup.n];
            for(int i=0;i<a.m;i++)
                for (int j=0;j<a.m;j++)
                    if (i==j)
                        inverse[i][j]=1;
                    else inverse[i][j]=0;
            BroomMatrix in=BroomMatrix.copy(b);
            float tempRow[]=new float[dup.n];

            for (int i=0;i<dup.n;i++)
                {

                    for (int j=i+1;j<dup.m;j++) // swap rows
                        {
                            if (Math.abs(dup.matrix[i][i])<Math.abs(dup.matrix[j][i]))
                                {
                                    dup=BroomMatrix.rowSwap(dup,i,j);
                                    in=BroomMatrix.rowSwap(in,i,j);
                                }



                        }//rows have been swapped

                    float pivot=dup.matrix[i][i];

                    if ((new Float(pivot)).isNaN()||(new Float(pivot)).isInfinite())
                        {
                            System.out.println("singular");
                            System.out.println(pivot);
```

```
                            return null;

                        }

                    //for each row under current, get a multiplier and apply it
                    for (int k=i+1;k<a.m;k++)
                        {
                            if(true)//dup.matrix[k][i]!=0)
                                {
                                    float m=dup.matrix[k][i]/pivot;

                                    for (int l=0;l<a.n;l++) //for each element in the
row
                                        {
                                            dup.matrix[k][l]=dup.matrix[k][l]-
m*dup.matrix[i][l];


                                        }

                                    for (int l=0;l<b.n;l++) //for each element in the
row
                                        {
                                            in.matrix[k][l]=in.matrix[k][l]-
m*in.matrix[i][l];
                                        }
                                }
                        }//now have an upper triangular matrix.

                }
            //System.out.println("HHHH");
            for (int i=a.m-1;i>=0;i--)
                {


                    float pivot2=dup.matrix[i][i];//get column pivot
                    if ((new Float(pivot2)).isNaN()||(new Float(pivot2)).isInfinite())
                        {
                            System.out.println("singular");
                            System.out.println(pivot2);
                            return null;
                        }


                    for (int j=i-1;j>=0;j--) //for each row above..
                        {
                            float m2=dup.matrix[j][i]/pivot2; //get multiplier for row

                            for (int k=0;k<a.n;k++)
                                {dup.matrix[j][k]=dup.matrix[j][k]-m2*dup.matrix[i][k];

                                }
                            for (int k=0;k<b.n;k++)
                                {
                                    in.matrix[j][k]=in.matrix[j][k]-m2*in.matrix[i][k];
                                }
                        }


                }
            //System.out.println("HHHH");

            for (int i=0;i<b.m;i++)
                for (int j=0;j<b.n;j++)
                    in.matrix[i][j]=in.matrix[i][j]/dup.matrix[i][i];

            return in;
        }
    else
        {
```

```
            // e("Gaussian elimination requires first argument to be a square matrix
and the second argument be a column vector with length equal to height of the first
arguement");


            if (a.m!=a.n)
                System.out.println("Input matrix (first parameter) to gaussian
elimination must be square");
            else if (a.m==a.n && a.m!=BroomMatrix.rank(a))
                {
                    System.out.println("Matrix: ");
                    BroomMatrix.printVariable(a);
                    System.out.println("is not full rank");
                    System.out.println("Input matrix (first parameter) to gaussian
elimination must be full rank.");
                }
            if (b.n!=1)
                {
                    System.out.println("B vector (second parameter) to gaussian
elimination must be an M x 1 matrix.  ");
                    BroomMatrix.printVariable(b);
                    System.out.println("is not M x 1, it has "+BroomMatrix.numCols(b)+"
columns.");
                }
            if (b.m!=a.m)
                {
                    System.out.println("The height (M) of the B vector must equal the
height (M)of the (A) input matrix.");
                    System.out.println("Here, (A) has "+ BroomMatrix.numRows(a)+" rows,
but (B) has "+ BroomMatrix.numRows(b)+" rows.");
                }
            e("");
            return null;
        }


    }


    /**
     * returns the tranpose of a matrix.
     *
     * @param a the matrix to be transposed
     */


    public static BroomMatrix trans(BroomMatrix a)
    {
        float temp[][]=new float[a.n][a.m];
        for (int i=0;i<a.m;i++)
            for (int j=0;j<a.n;j++)
                temp[j][i]=a.matrix[i][j];
        return new BroomMatrix(temp);
    }



    /**
     * returns the determinant of a matrix using the cofactor method
     *
     * @param a matrix to determinize
     *
     */

    public static float det(BroomMatrix a)
    {
        if (a.m==a.n)
            {
                float d=0;

            if (a.m==a.n && a.n==1)
```

```
            {//System.out.println("d is "+a.matrix[0][0]);
                return a.matrix[0][0];
            }
        else


            for (int i=0;i<a.n;i++)
                {
                    if (i%2==0)

d+=a.matrix[0][i]*BroomMatrix.det(BroomMatrix.getSubMatrix(a,0,i));
                    else
                        d-
=a.matrix[0][i]*BroomMatrix.det(BroomMatrix.getSubMatrix(a,0,i));
                }
            //System.out.println("d is "+d);
            return d;
            }
        else return 0;

    }

    /**
     * used in det(), this method will return the cofactor matrix , eliminating the row
and column of specified coordinate and returning the result
     *
     * @param a input matrix
     * @param b M coord
     * @param c N coord
     */

    public static BroomMatrix getSubMatrix(BroomMatrix a, int b, int c)
    {
        float temp[][]=new float[a.m-1][a.n-1];
        for (int i=0;i<a.m;i++)
            for (int j=0;j<a.m;j++)
                {
                    if (i<b)
                        {
                            if (j<c)
                                temp[i][j]=a.matrix[i][j];
                            if (j>c)
                                temp[i][j-1]=a.matrix[i][j];
                        }
                    if (i>b)
                        {
                            if (j<c)
                                temp[i-1][j]=a.matrix[i][j];
                            if (j>c)
                                temp[i-1][j-1]=a.matrix[i][j];
                        }
                }

        return new BroomMatrix(temp);
    }

    /**
     * returns the element at the specified location
     *
     * @param a source matrix
     * @param b1 M coord
     * @param c1 N coord
     *
     */

    public static float getElement(BroomMatrix a, float b1, float c1)
    {
        int b=BroomMatrix.round(b1);
        int c=BroomMatrix.round(c1);
```

```
        if(b<a.m && c<a.n && !(b<0) && !(c<0))
            return a.matrix[b][c];
        else
            {
                if(b<0 || c<0)
                    {

                        if(b1<0)
                            {
                                System.out.println("M= "+b);
                            }

                        if(c1<0)
                            {
                                System.out.println("N= "+c);
                            }
                        System.out.println("Matrix coordinates cannot be below zero");
                    }

                if(b>=a.m)
                    {
                        System.out.println("Requested M coordinate ("+b+") is out of range.
Height range is 0 to "+(a.m-1));
                    }
                if(c>=a.n)
                {
                    System.out.println("Requested N coordinate ("+c+") is out of range.
Width range is 0 to "+(a.n-1));
                }

                e("GetElement: Element requested out of matrix range");
                return 0;
            }
    }

    /**
     * returns the number of Columns in a matrix
     *
     * @param a source matrix
     *
     */

    public static int numCols(BroomMatrix a)
    {
        return a.n;
    }

    /**
     * returns the number of Rows in a matrix
     *
     * @param a source matrix
     *
     */
    public static int numRows(BroomMatrix a)
    {
        return a.m;
    }


    /**
     * returns a specified row of a matrix
     *
     * @param a source matrix
     * @param b1 Row Coord
     */

    public static BroomMatrix getRow(BroomMatrix a, float b1)
    {
        int b=BroomMatrix.round(b1);

        if (b<a.m && !(b<0)){
```

```
            BroomMatrix dup=BroomMatrix.copy(a);
            float temp[][]=new float[1][dup.n];
            temp[0]=dup.matrix[b];
            return new BroomMatrix(temp);
        }
        else
            {

                if (b<0)
                {
                    e("getRow: arguement cannot be less than zero.  Row "+b+" was
requested");
                    return null;
                }
                else
                    {
                        System.out.println("Matrix:");
                        BroomMatrix.printVariable(a);
                        System.out.println("Has "+a.m+" rows (numbered 0 to "+(a.m-1)+").
"+ b+" is out of that range.");
                        e("getRow: Argument out of bounds of matrix");
                        return null;
                    }
            }
    }

    /**
     * returns a specified column of a matrix
     *
     * @param a source matrix
     * @param b1 Column Coord
     */

    public static BroomMatrix getCol(BroomMatrix a, float b1)
    {
        int b=BroomMatrix.round(b1);


        if (b<a.n && !(b<0))
            {
                BroomMatrix dup=BroomMatrix.copy(a);
                float temp[][]=new float[dup.m][1];
                for (int i=0; i<dup.m; i++)
                    temp[i][0]=dup.matrix[i][b];
                return new BroomMatrix(temp);
            }
        else
            {

                if (b<0)
                    {
                        e("getCol: arguement cannot be less than zero.  Column "+b+" was
requested");
                        return null;
                    }
                else
                    {
                        System.out.println("Matrix:");
                        BroomMatrix.printVariable(a);
                        System.out.println("Has "+a.n+" columns (numbered 0 to "+(a.n-
1)+"). "+ b+" is out of that range.");
                        e("getCol: Argument out of bounds of matrix");
                        return null;
                    }
            }

    }

    /**
     * swaps rows of input matrix and returns the modified version of the matrix
     *
```

```
     * @param a source matrix
     * @param b1 first row to swap
     * @param c1 second row to swap
     */

    public static BroomMatrix rowSwap(BroomMatrix a, float b1, float c1)
    {
        int b=BroomMatrix.round(b1);
        int c=BroomMatrix.round(c1);
        if (b<a.m && c<a.m && b>=0 && c>=0)
            {
                BroomMatrix dup=BroomMatrix.copy(a);

                float temp[][]=dup.matrix;

                float tempRow[]=temp[b];
                temp[b]=temp[c];
                temp[c]=tempRow;
                return new BroomMatrix(temp);
            }
        else
            {
                {
                    if(b<0 || c<0)
                        {

                            if(b1<0)
                                {
                                    System.out.println("Row 1 for swap operation= "+b);
                                }

                            if(c1<0)
                                {
                                    System.out.println("Row 2 for swap operation= "+c);
                                }
                            System.out.println("Row indexes cannot be below zero");
                        }

                    if(b>=a.m)
                        {
                            System.out.println("First Requested row ("+b+") is out of
range. Height range is 0 to "+(a.m-1));
                        }
                    if(c>=a.m)
                        {
                            System.out.println("Second Requested row ("+c+") is out of
range. Height range is 0 to "+(a.m-1));
                        }


                }

                e("rowSwap: row swap out of range");
                return null;
            }


    }

    /**
     * swaps columns of input matrix and returns the modified version of the matrix
     *
     * @param a source matrix
     * @param b1 first column to swap
     * @param c1 second column to swap
     */
    public static BroomMatrix colSwap(BroomMatrix a,float b1, float c1)
    {
        int b=BroomMatrix.round(b1);
        int c=BroomMatrix.round(c1);
        if (b<a.n && c<a.n)
```

```
            {

                BroomMatrix dup=BroomMatrix.copy(a);
                float temp[][]=dup.matrix;
                float tempCol[]=new float[dup.m];


                for (int i=0;i<a.m;i++)
                    tempCol[i]=dup.matrix[i][b];

                for (int i=0;i<a.m;i++)
                    temp[i][b]=dup.matrix[i][c];


                for (int i=0;i<a.m;i++)
                    temp[i][c]=tempCol[i];
                return new BroomMatrix(temp);}
        else

            {
                {
                    if(b<0 || c<0)
                        {

                            if(b1<0)
                                {
                                    System.out.println("Column 1 for swap operation= "+b);
                                }

                            if(c1<0)
                                {
                                    System.out.println("Column 2 for swap operation= "+c);
                                }
                            System.out.println("Column indexes cannot be below zero");
                        }

                    if(b>=a.n)
                        {
                            System.out.println("First Requested Column ("+b+") is out of
range. Width  range is 0 to "+(a.n-1));
                        }
                    if(c>=a.n)
                        {
                            System.out.println("Second Requested Column ("+c+") is out of
range. Width range is 0 to "+(a.n-1));
                        }


                }

                e("rowSwap: Column swap out of range");
                return null;
            }

    }

    /**
     * adds two input matrices and returns the matrix sum
     *
     * @param a first matrix
     * @param b second matrix
     */
    public static BroomMatrix add(BroomMatrix a, BroomMatrix b)
    {
        if(a.m==b.m && a.n==b.n)
            {
                float temp[][]=new float[a.m][a.n];
                for (int i=0;i<a.m;i++)
                    {
                        for (int j=0;j<a.n;j++)
                            temp[i][j]=b.matrix[i][j]+a.matrix[i][j];
```

```
                }
            return new BroomMatrix(temp);
        }
       else
        {

            e("add: matrix sizes dont match\nMatrix A is an "+a.m+" by
"+a.n+"matrix,\nMatrix B is an "+b.m+" by "+b.n+"matrix");

            return null;
        }
    }

    /**
     * subtracts two input matrices and returns the matrix difference
     *
     * @param a first matrix
     * @param b second matrix
     */

    public static BroomMatrix subtract(BroomMatrix a, BroomMatrix b)
    {
       if(a.m==b.m && a.n==b.n)
        {
            float temp[][]=new float[a.m][a.n];
            for (int i=0;i<a.m;i++)
                {
                    for (int j=0;j<a.n;j++)
                        temp[i][j]=b.matrix[i][j]-a.matrix[i][j];
                }
            return new BroomMatrix(temp);
        }
       else
        {

            e("subtract: matrix sizes dont match\nMatrix A is an "+a.m+" by
"+a.n+"matrix,\nMatrix B is an "+b.m+" by "+b.n+"matrix");

            return null;
        }
    }



    /**
     * compares two floats and returns true or false on equality
     *
     * @param a first float
     * @param b second float
     *
     */
    public static boolean equals(float a,float  b)
    {
       if ((new Float(a)).equals(new Float(b)))
           return true;
       else return false;


    }

    /**
     * compares two matrices and returns true or false, based on a comparison of each
element in the matrices
     *
     * @param a first matrix
     * @param b second matrix
     *
     */
    public static boolean equals(BroomMatrix a,BroomMatrix b)
    {
```

```
        if(a.m==b.m && a.n==b.n)
            {
                boolean result=true;

                LOOP1: for (int i=0;i<a.m;i++)
                    LOOP2:  for (int j=0;j<a.n;j++)
                        if (!BroomMatrix.equals(a.matrix[i][j],b.matrix[i][j]))
                            {
                                result=false;
                                break LOOP1;
                            }
                return result;
            }
        else
            {

                return false;
            }
    }

    /**
     * compares two floats and returns true or false on inequality
     *
     * @param a first float
     * @param b second float
     *
     */
    public static boolean notEquals(float a,float  b)
    {
        if ((new Float(a)).equals(new Float(b)))
            return false;
        else return true;


    }

    /**
     * compares two matrices and returns true or false, based on a comparison of each
element in the matrices for inequality
     *
     * @param a first matrix
     * @param b second matrix
     *
     */
    public static boolean notEquals(BroomMatrix a,BroomMatrix b)
    {

        if(a.m==b.m && a.n==b.n)
            {
                boolean result=true;

                LOOP1: for (int i=0;i<a.m;i++)
                    LOOP2:  for (int j=0;j<a.n;j++)
                        if (!BroomMatrix.equals(a.matrix[i][j],b.matrix[i][j]))
                            {
                                result=false;
                                break LOOP1;
                            }
                return (!result);
            }

        else return true;

    }



    /**
     * subtracts two floats and returns difference
     *
```

```
 * @param a first float
 * @param b second float
 *
 */
public static float subtract(float a,float b)
{
    return a-b;
}


/**
 * adds two floats and returns sum
 *
 * @param a first float
 * @param b second float
 *
 */
public static float add(float a,float b)
{
    return a+b;
}



/**
 * multiplies two floats and returns the product
 *
 * @param a first float
 * @param b second float
 *
 */
public static float multiply(float a,float b)
{
    return a*b;
}



/**
 * multiplies a matrix by a scalar and returns the product
 *
 * @param a matrix
 * @param b scalar
 *
 */
public static BroomMatrix multiply(BroomMatrix a, float b)
{
    float temp[][]=new float[a.m][a.n];
    for (int i=0;i<a.m;i++)
        {
            for (int j=0;j<a.n;j++)
                temp[i][j]=b*a.matrix[i][j];
        }
    return new BroomMatrix(temp);

}

/**
 * multiplies a scalar  by a matrix and returns the product
 *
 * @param a scalar
 * @param b matrix
 *
 */
public static BroomMatrix multiply(float b, BroomMatrix a)
{
    float temp[][]=new float[a.m][a.n];
    for (int i=0;i<a.m;i++)
        {
            for (int j=0;j<a.n;j++)
```

```
                temp[i][j]=b*a.matrix[i][j];
            }
        return new BroomMatrix(temp);

    }

    /**
     * multiplies a matrix by a matrix and returns the product
     * order matters
     *
     * @param a matrix 1
     * @param b matrix 2
     *
     */

    public static BroomMatrix multiply(BroomMatrix a, BroomMatrix b)
    {


        if(a.n==b.m)
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                BroomMatrix dup2=BroomMatrix.copy(b);

                float result[][]=new float[dup1.m][dup2.n];

                for (int i=0;i<dup1.m;i++)

                    for (int j=0;j<dup2.n;j++)
                        {
                            result[i][j]=0;
                            for (int k=0;k<a.m;k++)
                                result[i][j]+=dup1.matrix[i][k]*dup2.matrix[k][j];
                        }


                return new BroomMatrix(result);
            }

        else
            {
                e("MatrixMultiplication: matrix sizes dont match\nMatrix A is an "+a.m+"
by "+a.n+"matrix,\nMatrix B is an "+b.m+" by "+b.n+"matrix.\nFor Matrix Multiplication
A*B, Width of A must match height of B");
                return null;
            }


    }




    /**
     *
     * prints a matrix object
     *
     * @param a matrix to print
     *
     */
    public static void printVariable(BroomMatrix a)
    {

        if (a!=null)
            {
                //System.out.println(a.m +" m rows, "+a.n+"n cols");
                System.out.println("");
                for (int i=0;i<a.m;i++)
                    {
                        for (int j=0;j<a.n;j++)
```

```
                        System.out.print(a.matrix[i][j]+" ");
                    System.out.println();
                }
            }
    }

    /**
     *
     * prints a float
     *
     * @param a float to print
     *
     */
    public static void printVariable(float a)
    {
        System.out.print(""+a+" ");
    }



    /**
     *
     * returns the rank of a matrix
     *
     * @param a matrix to evaluate
     *
     */



    public static int rank(BroomMatrix a)
    {
        BroomMatrix dup=BroomMatrix.copy(a);
        int lastP=-1;
        int rankNum=0;

        for (int i=0;i<a.m;i++) //for each row.
            {
                boolean found=false;
                for(int j=lastP+1;j<a.n;j++) //move to right on this row, from last known
pivot
                    {
                        if (!found)
                            {
                                for (int q=i+1;q<a.m;q++)

if(Math.abs(dup.matrix[i][j])<Math.abs(dup.matrix[q][j]))
                                        dup=BroomMatrix.rowSwap(dup,i,q);
                                //rows swapped


                                if (dup.matrix[i][j]!=0)//found pivot
                                    {
                                        found=true;
                                        rankNum++;
                                        lastP=j;
                                        float pivot=dup.matrix[i][j];

                                        for (int w=i+1;w<a.m;w++)//for each lower row, to
subtract
                                            {
                                                float m=dup.matrix[w][j]/pivot;
                                                for (int e=0;e<a.n;e++)
                                                    dup.matrix[w][e]=dup.matrix[w][e]-
m*dup.matrix[i][e];

                                            }

                                    }

                                else //no pivot
```

```
                                {}
                            }
                        }
                }
        return rankNum;

    }


    /**
     *
     * sets the specified entry of a matrix to the specified value, and returns the
modified matrix.
     *
     * @param a matrix
     * @param b value to set
     * @param m1 M coord
     * @param n1 N coord
     *
     */

    public static BroomMatrix setElement(BroomMatrix a, float b, float m1, float n1)
    {
        int m=BroomMatrix.round(m1);
        int n=BroomMatrix.round(n1);

        if ( m<a.m && n<a.n &&  m>=0 && n>=0 )
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                dup1.matrix[m][n]=b;
                return dup1;
            }
        else
            {
                if(m<0 || n<0)
                    {

                        if(m<0)
                            {
                                System.out.println("M= "+m);
                            }

                        if(n<0)
                            {
                                System.out.println("N= "+n);
                            }
                        System.out.println("Matrix coordinates cannot be below zero");
                    }

                if(m>=a.m)
                    {
                        System.out.println("Requested M coordinate ("+m+") is out of range.
Height range is 0 to "+(a.m-1));
                    }
                if(n>=a.n)
                    {
                        System.out.println("Requested N coordinate ("+n+") is out of range.
Width range is 0 to "+(a.n-1));
                    }

                e("setElement: Element requested out of matrix range");
                return null;
            }

    }

    /**
     *
     * sets the specified row of a matrix to the specified row vector, and returns the
modified matrix.
     *
```

```
     * @param a matrix
     * @param b row
     * @param pos1 M coord to set
     *
     */
    public static BroomMatrix setRow(BroomMatrix a, BroomMatrix b , float pos1)
    {
        int pos=BroomMatrix.round(pos1);



        if (pos<a.m && pos>=0 && b.m==1 && b.n==a.n)
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                for (int i=0;i<a.n;i++)
                    {
                        dup1.matrix[pos][i]=b.matrix[0][i];

                    }
                return dup1;
            }
        else
            {
                if (pos<0)
                    System.out.println("Row index ("+pos+") is less than 0");
                if (pos>=a.m)
                    System.out.println("Row index ("+pos+") is out of range.  Matrix
height range is 0 to "+(a.m-1));
                if (b.m!=1)
                    System.out.println("Matrix being used for setRow is not of height 1
(height = "+b.m +")");
                if (b.n!=a.n)
                    System.out.println("Row being used for setRow does not match width of
matrix");


                e("setRow: error");
                return null;
            }
    }




    /**
     *
     * sets the specified column of a matrix to the specified column vector, and returns
the modified matrix.
     *
     * @param a matrix
     * @param b column vector
     * @param pos1 N coord to set
     *
     */

    public static BroomMatrix setCol(BroomMatrix a, BroomMatrix b , float pos1)
    {
        int pos=BroomMatrix.round(pos1);
        if (pos<a.n && b.n==1 && pos>=0 && b.m==a.m)
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                for (int i=0;i<a.m;i++)
                    {
                        dup1.matrix[i][pos]=b.matrix[i][0];

                    }

                return dup1;
            }
        else
            {
```

```
                if (pos<0)
                    System.out.println("Column index ("+pos+") is less than 0");
                if (pos>=a.n)
                    System.out.println("Column index ("+pos+") is out of range.  Matrix
width range is 0 to "+(a.n-1));
                if (b.n!=1)
                    System.out.println("Matrix being used for setCol is not of width 1
(width = "+b.m +")");
                if (b.m!=a.m)
                    System.out.println("Matrix being used for setCol does not match height
of matrix");


                e("setCol: error");
                return null;
            }

    }

    /**
     *
     * adds the specified row to end of the matrix, and returns the modified matrix.
     *
     * @param a matrix
     * @param b row vector
     *
     */

    public static BroomMatrix addRow(BroomMatrix a, BroomMatrix b)
    {
        if (a.n==b.n && b.m==1)
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                BroomMatrix dup2=BroomMatrix.copy(b);
                return insertRow(dup1,dup2, (float)dup1.m);
            }

        else
            {
                if (a.n!=b.n)
                    System.out.println("addRow: Matrix and Row being added must be the
same width.\nMatrix width="+a.n+", row width=+"+b.n);
                if (b.m==1)
                    System.out.println("addRow: row being added must have height 1, Row
height="+b.m);
                e("addRow:Error");
                return null;
            }

    }

    /**
     *
     * adds the specified column to the right of the  matrix, and returns the modified
matrix.
     *
     * @param a matrix
     * @param b row vector
     *
     */
    public static BroomMatrix addCol(BroomMatrix a, BroomMatrix b)
    {
        if (a.m==b.m && b.n==1)
            {
                BroomMatrix dup1=BroomMatrix.copy(a);
                BroomMatrix dup2=BroomMatrix.copy(b);
                return insertCol(dup1,dup2, (float)dup1.n);
            }

        else
            {
```

```
            if (a.m!=b.m)
                System.out.println("addCol: Matrix and Column being added must be the
same height.\nMatrix height="+a.m+", column height=+"+b.m);
            if (b.n==1)
                System.out.println("addCol: col being added must have width 1, column
width="+b.n);
            e("addCol:Error");
          return null;
          }
    }


    /**
     *
     * adds the specified row(s) into the matrix at the specified loccation, shifting the
row in that position and all rows below it, down - returns modified matrix.
     *
     * @param a matrix
     * @param b row(s)
     * @param pos1 location to insert
     *
     */


    public static BroomMatrix insertRow(BroomMatrix a, BroomMatrix b, float pos1)
    {

       int pos=BroomMatrix.round(pos1);

       if (b.n==a.n)
           {
              float temp[][]=new float[a.m+b.m][a.n];

              for (int i=0;i<pos;i++)
                  for (int j=0;j<a.n;j++)
                     {

                        temp[i][j]=a.matrix[i][j];

                     }

              for (int i=0;i<b.m;i++)
                  for (int j=0;j<a.n;j++)
                     {
                        temp[i+pos][j]=b.matrix[i][j];

                     }

              for (int i=pos;i<a.m;i++)
                  for (int j=0;j<a.n;j++)
                     {
                        temp[i+b.m][j]=a.matrix[i][j];


                     }
              return (new BroomMatrix(temp));
           }
       else
           {
              e("row widths do not match");
              return null;
           }


    }


    /**
     *
```

```
     * adds the specified columns(s) into the matrix at the specified loccation, shifting
the column in that position and all columns to the right of  it, to the right - returns
modified matrix.
     *
     * @param a matrix
     * @param b column(s)
     * @param pos1 location to insert
     *
     */
    public static BroomMatrix insertCol(BroomMatrix a, BroomMatrix b, float  pos1)
    {
       int pos=BroomMatrix.round(pos1);

       if (a.m==b.m)
           {
              float temp[][]=new float[a.m][a.n+b.n];



              for (int i=0;i<pos;i++)
                  for (int j=0;j<a.m;j++)
                     {

                        temp[j][i]=a.matrix[j][i];

                     }


              for (int i=0;i<b.n;i++)
                  for (int j=0;j<b.m;j++)
                     {

                        temp[j][i+pos]=b.matrix[j][i];

                     }

              for (int i=pos;i<a.n;i++)
                  for (int j=0;j<a.m;j++)
                     {

                        temp[j][i+b.n]=a.matrix[j][i];

                     }




              return (new BroomMatrix(temp));
           }


       else {e("col widths do not match");
       return null;}


    }



    /**
     *
     * removes the specified row from the input matrix and returns modified matrix.
     * (shifts rows up)
     *
     * @param a matrix
     * @param pos1 position to delete
     *
     */
    public static BroomMatrix deleteRow(BroomMatrix a, float pos1)
    {
       int pos=BroomMatrix.round(pos1);
```

```
        if (pos<a.m && pos>=0)
            {
                //System.out.println("delete begin");


                float temp[][]=new float[a.m-1][a.n];


                BroomMatrix tempRes=new BroomMatrix(temp);
                BroomMatrix q=null;
                for (int i=0;i<pos;i++)
                    {
                        tempRes=BroomMatrix.setRow(tempRes,BroomMatrix.getRow(a,i),i);
                    }


                for (int i=pos+1;i<a.m;i++)
                    {
                        tempRes=BroomMatrix.setRow(tempRes,BroomMatrix.getRow(a,i),i-1);

                    }

                //System.out.println("delete end ");
                return tempRes;
            }
        else
            {
                if (pos<0)
                    {
                        System.out.println("Row index ("+pos+") cannot be below zero");
                    }
                if (pos>=a.m)
                    {
                        System.out.println("Row index ("+pos+") out of range.  valid Range
is 0 to "+(a.m-1));
                    }

                e("deleteRow: row index out of range");
                return null;
            }



    }




    /**
     *
     * removes the specified column from the input matrix and returns modified matrix.
     * (shifts rows left)
     *
     * @param a matrix
     * @param pos1 position to delete
     *
     */
    public static BroomMatrix deleteCol(BroomMatrix a, float pos1)
    {
        int pos=BroomMatrix.round(pos1);
        if (pos<a.n  && pos>=0)
            {
                System.out.println("delete begin");


                float temp[][]=new float[a.m][a.n-1];


            BroomMatrix tempRes=new BroomMatrix(temp);
```

```
            for (int i=0;i<pos;i++)
                {
                    tempRes=BroomMatrix.setCol(tempRes,BroomMatrix.getCol(a,i),i);
                }


            for (int i=pos+1;i<a.n;i++)
                {
                    tempRes=BroomMatrix.setCol(tempRes,BroomMatrix.getCol(a,i),i-1);
                }

            System.out.println("delete end ");
            return tempRes;
            }
        else
            {
               if (pos<0){System.out.println("Col index ("+pos+") cannot be below
zero");}
               if ( pos>=a.n){System.out.println("Column index ("+pos+") out of range.
valid Range is 0 to "+(a.n-1));}

               e("deleteCol: Column index out of range");
               return null;
            }



    }







    /**
     *
     * appends second matrix to right of first, and returns the concatenation
     *
     * @param a matrix 1
     * @param b matrix 2
     *
     */
    public static BroomMatrix appendLR(BroomMatrix a, BroomMatrix b)
    {
        if (a.m==b.m)
            {
               BroomMatrix dup1=BroomMatrix.copy(a);
               BroomMatrix dup2=BroomMatrix.copy(b);
               return insertCol(dup1,dup2, dup1.n);
            }
        else
            {
               e("appendLR:matrix heights dont match.\n Matrix A height="+a.m+", Matrix b
height="+b.m);
               return null;
            }
    }

    /**
     *
     * appends second matrix to bottom of first, and returns the concatenation
     *
     * @param a matrix 1
     * @param b matrix 2
     *
     */
    public static BroomMatrix appendUD(BroomMatrix a, BroomMatrix b)
    {
        if (a.n==b.n)
            {
```

```
                BroomMatrix dup1=BroomMatrix.copy(a);
                BroomMatrix dup2=BroomMatrix.copy(b);
                return insertRow(dup1,dup2, dup1.m);
            }
        else
            {
                e("appendUD:matrix widths dont match.\n Matrix A width="+a.n+", Matrix b
width="+b.n);
                return null;
            }
    }



    public static void e(String s) {

        System.out.println(s);
        System.exit(1);

    }

    public static int round(float a)
    {
        int q=(int)a;
        if (a-q<.5)
            return q;
        else return q+1;

    }

}
```

---

```
total revisions: 12;  selected revisions: 12
description:
added basic error messages,
changed args to floats for user input
(this was modified from start.java to remove the test main method)
----------------------------
revision 1.12
date: 2003/05/12 18:26:41;  author: mjw133;  state: Exp;  lines: +0 -53
Remoed some testing code that was comment out in the file.
----------------------------
revision 1.11
date: 2003/05/12 14:05:34;  author: gg;  state: Exp;  lines: +1 -1
fixed setelment error check
----------------------------
revision 1.10
date: 2003/05/11 06:44:18;  author: gg;  state: Exp;  lines: +1382 -1030
javaDoc'ed
----------------------------
revision 1.9
date: 2003/05/09 19:00:29;  author: gg;  state: Exp;  lines: +56 -21
more error messages.
,
----------------------------
revision 1.8
date: 2003/05/09 18:29:36;  author: gg;  state: Exp;  lines: +163 -33
more error stuff.
----------------------------
revision 1.7
date: 2003/05/09 17:29:34;  author: gg;  state: Exp;  lines: +151 -61
added more robust error messages
----------------------------
revision 1.6
date: 2003/05/05 01:44:27;  author: gg;  state: Exp;  lines: +2 -2
add constraint on setRow/Col.
```

```
----------------------------
revision 1.5
date: 2003/05/05 01:30:40;  author: gg;  state: Exp;  lines: +22 -5
added matrix subtraction
----------------------------
revision 1.4
date: 2003/05/05 01:24:53;  author: gg;  state: Exp;  lines: +1 -1
fixed colSwap parameters (ints--> floats)
----------------------------
revision 1.3
date: 2003/05/04 23:46:14;  author: gg;  state: Exp;  lines: +12 -0
add newMatrix function (zero matrix).
----------------------------
revision 1.2
date: 2003/05/02 20:40:04;  author: gg;  state: Exp;  lines: +51 -75
fixed append UD compare and removed creatio.
----------------------------
revision 1.1
date: 2003/05/02 20:27:07;  author: gg;  state: Exp;
Initial revision
=============================================================================
```

---

And now for the test cases:

---

HelloWorld

```
//
// A simple program to test if the parser can handle
// a really simple case.
//
// Author: Michael Weiss
//
//

start()

        print("Hello World!");

endstart
```

---

ControlMadness

```
// A Test Case program
//
// has nested control flow variables
// for testing control flow parameters
//
// Author: Michael Weiss
//

define function Void runForAWhile()

        Number x,y,z;

        x = 0;
        z = 0;

        while ( x < 10 )

                for ( y = 0; y < 10; y=y+1; )
```

```
                        z = z+1;
                endfor

                x = x + 1;

        endwhile

        print("working program should print 100");
        z.print();

endfunction


start()
        runForAWhile();
endstart
```

---

ExperimentBroom

```
// ExperimentBroom
//
// Author: Michael Weiss
//
// A test suite program.
// This is a very simple program written to
// test the case of a programmer trying to figure out what all
// of the different built-in functions in broom do, by plugging in hard-coded
// values.
//
// He just wants to see if the output compares to what he expects
//
// The purpose of this is to test all of the semantic checking
// on the builtin functions to make sure that it does not throw
// errors when it shouldn't
//

start()
        Number rn;
        Matrix rm;

        rm = [1,2;3,4;];
        rm.print();

        rm = inv([1,2;3,4;]);
        print("inv:");
        rm.print();

        rn = det([1,2;3,4;]);
        print("det:");
        rn.print();

        rm = trans( [1,2;3,4;] );
        print("trans:");
        rm.print();

        rn = rank( [1,2;3,4;] );
        print("rank:");
        rn.print();

        rn = numRows( [1,2;3,4;] );
        print("numrows");
        rn.print();

        rn = numCols ( [1,2;3,4;] );
        print("numcols");
        rn.print();

        rm = gauss( [1,2;3,4;], [1;0;]);
        print("gauss");
```

```
        rm.print();

        rm = getRow( [1,2;3,4;], 0);
        rm = getRow( [1,2;3,4;], 1);

        rm = getCol( [1,2;3,4;], 0);
        rm = getCol( [1,2;3,4;], 1);

        print("col 1");
        rm.print();

        rm = appendLR( [1,2;3,4;], [1,2;3,4;] );
        print("appendLR: ");
        rm.print();

        rm = appendUD( [1,2;3,4;], [1,2;3,4;] );

        print("appendUD: ");
        rm.print();

        rm = deleteRow( [1,2;3,4;], 1 );
        rm = deleteRow( [1,2;3,4;], 0 );

        rm = deleteCol( [1,2;3,4;], 0 );
        rm = deleteCol( [1,2;3,4;], 1 );

        print("del col 2");
        rm.print();

        //programmer: is scalar multiplication allowed?
        //  let's try it!!!

        rm = [1,2;3,4;] * 2;
        print("times 2");
        rm.print();

        rn = getElement( [1,2;3,4;], 0, 0);
        rn = getElement( [1,2;3,4;], 1, 1);

        print("expect 4:");
        rn.print();

        rm = rowSwap( [1,2;3,4;], 0, 1 );
        print("rowswap: ");
        rm.print();

        rm = colSwap( [1,2;3,4;], 0, 1 );
        print("colswap: ");
        rm.print();

        rm = setRow( [1,2;3,4;], [8,8;], 0 );

        print("set top row to 8,8");
        rm.print();

        rm = setCol( [1,2;3,4;], [8;8;], 0 );

        print("set left column to 8,8");
        rm.print();

        rm = setElement( [1,2;3,4;], 0, 0, 0 );
        print("set upper-left element to 0: ");
        rm.print();

        print("that is all.");

endstart

-------------------------------------------------------------------------------

Gauss
```

```
// A simple program to test the functionality
// of Gaussian elimination
//
// by Michael Weiss
//

global Matrix A,b;

A = [ 1, 2;
      3, 5; ];

b = [6;7;];


start()

        Matrix Ans;

        Ans = gauss(A,b);

        //print results
        print("results of gaussian elimination:");

        Ans.print();

endstart

--------------------------------------------------------------------------------

ALU

// slu square factorization with no row exchanges
//
// some slu function is code translation from Linear Algebra, by Gilbert Strang
//
//
// all other code and code translation is by: Michael Weiss
//
// supplementary functions: Michael Weiss
//
// a rigorous test case for our language.
//

global Matrix X;
global Number iterations, tol;
global Number TRUE, FALSE;

X = [ 1, 1, 0, 0;
      1, 2, 1, 0;
      0, 1, 2, 1;
      0, 0, 1, 1; ];

TRUE = 1;
FALSE = 0;


tol = 1.0e-6;

//returns the absolute value of number
define function Number absol( Number x )

        if ( x > 0 ) then
                return x;
        else
                return (x * -1);
        endif

endfunction

//returns TRUE if matrix A is square and 0 if it is not
```

```
define function Number isSquare( Matrix A )

        Number m,n;

        m = numRows(A);
        n = numCols(A);

        if ( m == n ) then
                return TRUE;
        else
                return FALSE;
        endif

endfunction

define function Matrix copy(Matrix A)
        return A;
endfunction

// returns an empty matrix of the same size C
define function Matrix newEmptyMatrix (Matrix C)

        Matrix R;
        Number i,k;

        R = copy(C);

        for ( i=0; i < numRows(R) ; i=i+1; )
                for ( k=0; k < numCols(R); k=k+1; )
                        R = setElement(R,0,i,k);
                endfor
        endfor

        return R;

endfunction

// this function assumes that A is a square matrix or
// it will die.
define function Matrix makeIdentity (Matrix A)

        Number i,j,n;

        n = numRows(A);

        for ( i=0; i<n; i=i+1; )
                for ( j=0; j<n; j=j+1; )

                        if ( i == j ) then
                                A = setElement(A,1,i,j);
                        else
                                A = setElement(A,0,i,j);
                        endif

                endfor
        endfor

        return A;

endfunction


define function Matrix slu( Matrix A )

        //define all local variables
        Matrix P, LU;
        Number i,j,k,r;
        Number n;
        Number maxRow;

        P = newEmptyMatrix(A);
```

```
        //convert P into an identity matrix
        P = makeIdentity( P );
        //this is the permutation matrix

        //LU contains the lower and upper factorization of the matrix
        //but it begins as an empty matrix
        LU = newMatrix( numRows(A) ,numCols(A) );

        // die if A is not square
        if ( ! (isSquare( A )==TRUE) ) then
               print ("error matrix A on slu must be square!");
               return A;
        endif

        n = numRows(A);
        //print("number of rows below"); n.print(); print("");

        for (k=0;k<n;k=k+1;)

               //choose the largest number in row k or below.
               //exchange its row with k.
               maxRow = k;
               for (r=k+1;r<n;r=r+1;)

                        if ( absol(getElement(A,r,k)) > (absol(getElement(A,maxRow,k))) )
then
                                maxRow = r;
                        endif

              endfor

               if (maxRow != k) then

                A  = rowSwap(A,k,maxRow);
                LU = rowSwap(LU,k,maxRow);
                P  = rowSwap(P,k,maxRow);

               endif

               for (i=k+1;i<n;i=i+1;)

                        LU = setElement(LU,  getElement(A,i,k)/getElement(A,k,k)  ,i,k);

                         for (j=k+1;j<n;j=j+1;)

                                A = setElement(A, getElement(A,i,j) -
getElement(LU,i,k)*getElement(A,k,j)  ,i,j);

                         endfor

                endfor

                 for (j=k;j<n;j=j+1;)

                        LU = setElement(LU, getElement(A,k,j),  k,j);

                endfor

        endfor

        return LU;

endfunction

start()

        Matrix LU;
        Matrix L, U;
        Matrix W;
```

```
        Number i,j;

        print("performing factorization on this matrix: ");
        X.print();
        print("\n\n");

        LU = slu(X);

        print("The result of the factorization: \n\n");

        print("LU:");

        LU.print();

        L = newMatrix(numRows(LU),numCols(LU));
        U = copy(L);

        //get the L part of the matrix
        for ( i=0; i<numRows(LU); i=i+1; )
                for ( j=0; j<numRows(LU); j=j+1; )

                        //if you are looking at the lower half of the matrix
                        if (i > j) then
                                L = setElement(L,getElement(LU,i,j),i,j);
                        endif

                        if (i < j || i == j) then
                                U = setElement(U,getElement(LU,i,j),i,j);
                        endif

                        if (i==j) then
                                L = setElement(L,1,i,j);
                        endif

                endfor
        endfor

        print("L:");
        L.print();

        print("U:");
        U.print();

        print("L * U, returns the original matrix:");

        W = L * U;

        W.print();

endstart

--------------------------------------------------------------------------------

Identity


//
// Here's a useful function that computes the idenity
// matrix for a bunch of functions
//
define function Matrix I( Number m, Number n )

        Number j,k;
        Matrix Id, TempColumn;

        Id = [1;];
        //first build Id into a column of hieght m
        for ( j=0; j<(m-1) ; j=j+1; )
                Id = appendUD( Id, [0;] );
        endfor
```

```
        // now Id is a column of height m
        Id.print(); //for debugging

        //builds the matrix is a difficult fashion
        for ( k=0; k<n-1; k=k+1; )
                TempColumn = [0;];
                //build up another column
                for ( j=0; j<(m-1) ; j=j+1; )
                        if ( j == k ) then
                                TempColumn = appendUD( TempColumn, [1;] );
                        else
                                TempColumn = appendUD( TempColumn, [0;] );
                        endif
                endfor

                Id = appendLR( Id, TempColumn );
        endfor

        //Identity matrix has been constructed
        Id.print();

        return Id;

endfunction


start()

        Matrix I1, I2, I3, I4,I8;

        //go through and declare each one of them
        I1 = I(1,1);
        I1.print();

        I2 = I(2,2);
        I2.print();

        I3 = I(3,3);
        I3.print();

        I4 = I(4,4);
        I4.print();

        I8 = I(8,8);
        I8.print();

        print("done");

endstart

================================================================================

Statistics

// test case by Michael Weiss
//
// this program computes the statistics
// which are stored in a matrix
//

global Matrix data1,data2;

data1 = [ 1,2,3;
          4,5,6;
          7,8,9; ];

data2 = [ 0,1,0;
          1,0,1;
          1,1,1; ];

define function Number findAverage( Matrix d )
```

```
        Number i,j;
        Number m,n,avg,sum,num;

        m = 1; n = 1;
        m = numRows( d );
        n = numCols( d );

        num = m*n;

        sum = 0;

        for ( i=0; i<m; i=i+1; )
                for (j=0; j<n; j=j+1; )
                        sum = sum + getElement(d,i,j);
                endfor
        endfor

        avg = sum / num;

        return avg;

endfunction

start()

        Number a,b;

        a = findAverage( data1 );

        print("average:");
        a.print();
        print("");

endstart
```

---

RESULTS TO TESTING THE FILES (By Michael Weiss) (Examples of developing a broom program)

2 Examples of programs in BROOM

---

ALU program  -- working example of broom progam (ALU.broom)

---

ALU.broom  -- development phase of the sample program along with output from the compiler:

```
[mjw133@aluminor plt-project]$ java BCC /proj/test_cases/ALU.broom
/proj/test_cases/ALU.broom
line 81:39: expecting SEMI, found ')'
line 83:25: expecting RPAREN, found 'if'
line 83:30: unexpected token: i
line 83:35: unexpected token: j
line 85:25: expecting "endfor", found 'else'
line 87:25: expecting "endfunction", found 'endif'
line 130:27: expecting SEMI, found ')'
line 134:15: expecting RPAREN, found 'maxRow'
line 134:24: unexpected token: k
line 138:33: expecting "then", found 'maxRow'
line 138:42: unexpected token: r
line 139:25: expecting "endfor", found 'endif'
line 141:15: expecting "endfunction", found 'endfor'
BroomProgram did not compile; the following errors were detected:
* Error in start() : reference to undefined function slu
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.
* Error in start() : reference to undefined function slu
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.
```

```
after the initial compilation:

java BCC /proj/test_cases/ALU.broom
/proj/test_cases/ALU.broom
BroomProgram did not compile; the following errors were detected:
* Error in slu : function call with incorrect number of arguments.
        makeIdentity expected 1, but got 2.
* Error in slu : reference to undefined function selElement
* Error in slu : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.


java BCC /proj/test_cases/ALU.broom
/proj/test_cases/ALU.broom
BroomProgram did not compile; the following errors were detected:
* Error in start() : trying to print undefined variable L
* Error in start() : trying to print undefined variable U
* Error in start() : reference to undefiend variable L
* Error in start() : reference to undefiend variable U
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.



[mjw133@aluminor plt-project]$ java BCC /proj/test_cases/ALU.broom
/proj/test_cases/ALU.broom
[mjw133@aluminor plt-project]$ javac ALU.java
[mjw133@aluminor plt-project]$ java ALU
performing factorization on this matrix:

1.0 1.0 0.0 0.0
1.0 2.0 1.0 0.0
0.0 1.0 2.0 1.0
0.0 0.0 1.0 1.0



The result of the factorization:


LU:

1.0 1.0 0.0 0.0
1.0 1.0 1.0 0.0
0.0 1.0 1.0 1.0
0.0 0.0 1.0 0.0
L:

1.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0
0.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0
U:

1.0 1.0 0.0 0.0
0.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0
0.0 0.0 0.0 0.0
L * U, returns the original matrix:

1.0 1.0 0.0 0.0
1.0 2.0 1.0 0.0
0.0 1.0 2.0 1.0
0.0 0.0 1.0 1.0
[mjw133@aluminor plt-project]$


--------------------------------------------------------------------------------
Statistics -- Average Finder  (Stat2.broom)
```

```
--------------------------------------------------------------------------------

java BCC /proj/test_cases/Stat2.broom
/proj/test_cases/Stat2.broom

[mjw133@aluminor plt-project]$ javac Statistics.java
[mjw133@aluminor plt-project]$ java Statistics
average:
5.0
[mjw133@aluminor plt-project]$

--------------------------------------------------------------------------------
ControlMadness -- tests some control loop implementations (ControlMadness.broom)
--------------------------------------------------------------------------------

java BCC /proj/test_cases/ControlMadness.broom
/proj/test_cases/ControlMadness.broom
[mjw133@aluminor plt-project]$ javac ControlMadness.java
[mjw133@aluminor plt-project]$ java ControlMadness
working program should print 100
100.0 [mjw133@aluminor plt-project]$

--------------------------------------------------------------------------------
Gauss - development and test case (Gauss.broom)
--------------------------------------------------------------------------------

Before corrections were made, a runtime error occured, which allowed
for easy debugging:

java BCC /proj/test_cases/Gauss.broom
/proj/test_cases/Gauss.broom
[mjw133@aluminor plt-project]$ javac Gauss.java
[mjw133@aluminor plt-project]$ java Gauss
B vector (second parameter) to gaussian elimination must be an M x 1 matrix.

6.0 7.0
is not M x 1, it has 2 columns.
The height (M) of the B vector must equal the height (M)of the (A) input matrix.
Here, (A) has 2 rows, but (B) has 1 rows.

[mjw133@aluminor plt-project]$

After correcting the error:

java Gauss
results of gaussian elimination:

-16.000004
11.000002
[mjw133@aluminor plt-project]$


--------------------------------------------------------------------------------
ExperimentBroom.broom
--------------------------------------------------------------------------------
java BCC /proj/test_cases/ExperimentBroom.broom
/proj/test_cases/ExperimentBroom.broom
[mjw133@aluminor plt-project]$ javac ExperimentBroom.java
[mjw133@aluminor plt-project]$ java ExperimentBroom

1.0 2.0
3.0 4.0
inv:

-2.0000002 1.0000001
1.5000001 -0.50000006
det:
-2.0 trans:

1.0 3.0
2.0 4.0
```

```
rank:
2.0 numrows
2.0 numcols
2.0 gauss

-2.0000002
1.5000001
col 1

2.0
4.0
appendLR:

1.0 2.0 1.0 2.0
3.0 4.0 3.0 4.0
appendUD:

1.0 2.0
3.0 4.0
1.0 2.0
3.0 4.0
delete begin
delete end
delete begin
delete end
del col 2

1.0
3.0
times 2

2.0 4.0
6.0 8.0
expect 4:
4.0 rowswap:

3.0 4.0
1.0 2.0
colswap:

2.0 1.0
4.0 3.0
set top row to 8,8

8.0 8.0
3.0 4.0
set left column to 8,8

8.0 2.0
8.0 4.0
set upper-left element to 0:

0.0 2.0
3.0 4.0
that is all.
[mjw133@aluminor plt-project]$


-------------------------------------------------------------------------------
HelloWorld   example program      (HelloWorld.broom)
-------------------------------------------------------------------------------

java BCC /proj/test_cases/HelloWorld.broom
/proj/test_cases/HelloWorld.broom
[mjw133@aluminor plt-project]$ javac HelloWorld.java
[mjw133@aluminor plt-project]$ java HelloWorld
Hello World!
[mjw133@aluminor plt-project]$

-------------------------------------------------------------------------------
IncrementalOperations  (early testing phase)           (MatrixFun.broom)
```

```
--------------------------------------------------------------------------------


[mjw133@aluminor plt-project]$ java BCC /proj/test_cases/MatrixFun.broom
/proj/test_cases/MatrixFun.broom
[mjw133@aluminor plt-project]$ javac IncrementalOperations.java
[mjw133@aluminor plt-project]$ java IncrementalOperations
if this program is working, it should print 2....
2.0

a is square, returning determinant.

if this program is working, it should print 8...
8.0 a is not square, returning 0
if this program is working, it should print 0 ....
0.0


about the increment the matrix:

1.0 1.0
1.0 1.0
 each element by:
5.0



 Matrix after incrementing:

6.0 6.0
6.0 6.0
if this program is working, it should print a matrix of
 all 6's that is 2x2

6.0 6.0
6.0 6.0
[mjw133@aluminor plt-project]$


--------------------------------------------------------------------------------
The following are test cases for error our programs error detection
Designed by Brian Pellegrini
--------------------------------------------------------------------------------

logicalOpps

// Test for logical operator errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//use of logical operators works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1;];
B= [0, 1; 1, 0;];


start()
        Number test;
        test = 0;

        if (A > B) then
                print (" > checker didn't work");
        endif

        if (A < B) then
                print (" < checker didn't work");
        endif
```

```
        if (A == test) then
                print (" LOGEQ type checker didn't work");
        endif

        if (A != test) then
                print (" LOGNEQ type checker didn't work");
        endif
endstart


--------------------------------------------------------------------------------
Results for logicalOpps.broom test:

BroomProgram did not compile; the following errors were detected:
* Error in start() : '>' operator cannot be applied to matrices.
* Error in start() : '<' operator cannot be applied to matrices.

* Error in start() : mismatched types on '==' logical expression.

* Error in start() : mismatched types on '!=' logical expression.
--------------------------------------------------------------------------------

noaryOp

// Test for noary operations checks
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//noary operations works. it checks
//DET, RANK, NUMROWS, NUMCOLS, INV, TRANS
// It checks the type checking and the number
// of arguments checking.

// it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1;];


//didn't catch the type mismatch errors
start()
        Number q;

        B = det(q);
        q = trans(A);
        B = rank();

endstart

--------------------------------------------------------------------------------

Results for noary opperations test


line 26:18: unexpected token: )
line 26:19: expecting RPAREN, found ';'
line 30:1: unexpected token: null
line 30:1: unexpected token: null
line 30:1: expecting SEMI, found 'null'
line 30:1: expecting "endstart", found 'null'
<AST>: expecting START, found '<ASTNULL>'
<AST>: expecting START, found '<ASTNULL>'
<AST>: expecting START, found '<ASTNULL>'

--------------------------------------------------------------------------------
NOTE FROM THE AUTHOR OF STATIC CHECKER: This program did not show static
errors because it is syntactically incorrect.
--------------------------------------------------------------------------------
```

```
reservedWord

// Test for reserved word error checking
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//use of reserved words works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix throws;


define function Void implements ()
        print ("didn't work");
endfunction

start()
        Number Matrix;

        Number = implements();
endstart

--------------------------------------------------------------------------------
Results for reserved word test:


BroomProgram did not compile; the following errors were detected:
* Error in *global declaration* : The word throws is a reserved java keyword.
* Error in *global declaration* : The word implements is a reserved java keyword.
* Error in implements : The word implements is a reserved java keyword.
* Error in start() : The word implements is a reserved java keyword.

--------------------------------------------------------------------------------

returnNonVoid

// Test for returning a value in a void
//function
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//trying to return a value for a void function
//...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;



A= [1, 0; 0, 1;];
B= [0, 1; 0 , 2;];


define function Void myFunction ()
        Number test = 0;
        return test;
endfunction

start()
        Number test1;
        test1 = myFunction;
endstart

--------------------------------------------------------------------------------

These are the results for the test of a void function
trying to return a value:

BroomProgram did not compile; the following errors were detected:
* Error in myFunction : reference to undefiend variable test
* Error in myFunction : return statement in function of 'Void' return
```

```
type.
         Please remove the return statement or give function return type.
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are
operational

--------------------------------------------------------------------------------

setRowCol

// Test for row swap, col swap, and element
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//assignment of row and column values works
//values works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B,C;

A= [1, 0; 0, 1;];
B= [0, 1; 0 , 2;];
C= [1, 1; 1, 1;];



//didn't catch the type mismatch errors
start()
        Number myEl;

        A = rowSwap (A, 5, 8);
        B = colSwap (A, B, 6);
        myEl = getElement(A, B, 12);
        C = rowSwap (A, 1, 1, 1);



endstart

--------------------------------------------------------------------------------

Results for row and column swap test:


BroomProgram did not compile; the following errors were detected:
* Error in start() : on param 2, colSwap requires MATRIX, NUMBER, NUMBER as arguments.
* Error in start() : on param 2, getElement requires MATRIX, NUMBER, NUMBER as arguments.

--------------------------------------------------------------------------------

runTimeTest

// Run Time errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//run-time errors works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1; 2,0;];
B= [0, 1; 1, 0;];


start()
        //row number is out of bounds
        //should cause a run time error
        A = setElement(A, 5, 1, 10);
```

```
endstart

--------------------------------------------------------------------------------


Results of run time test for out of bounds error:


Requested N coordinate (10) is out of range. Width range is 0 to 1
setElement: Element requested out of matrix range

--------------------------------------------------------------------------------

runTimeTest2

// Run Time errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//run-time errors works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1; 2,0;];
B= [0, 1; 1, 0;];


start()
        //matrices have different heights
        //should cause a run time error
        A = gauss(A, B);


endstart

--------------------------------------------------------------------------------

Run-time test for improperly formatted matrices:

Input matrix (first parameter) to gaussian elimination must be square
B vector (second parameter) to gaussian elimination must be an M x 1 matrix.

0.0 1.0
1.0 0.0
is not M x 1, it has 2 columns.
The height (M) of the B vector must equal the height (M)of the (A) input matrix.
Here, (A) has 3 rows, but (B) has 2 rows.


--------------------------------------------------------------------------------



--------------------------------------------------------------------------------

setRowCol

// Test for set row and set cols errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//assignment of row and column values works
//values works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0; 0, 1;];
B= [0, 1;0 , 2;];
```

```
start()
        A = setRow (A, 5, [1, 2;]);
        A = setCol (5, B, 2);
        B = setRow (A, [1, 2, 3;], 1);
endstart
```

---

Results for setRow and setCol:

```
BroomProgram did not compile; the following errors were detected:
* Error in start() : on param 2, setRow requires MATRIX, MATRIX, NUMBER as arguments.
* Error in start() : on param 3, setRow requires MATRIX, MATRIX, NUMBER as arguments.
* Error in start() : on param 1, setCol requires MATRIX, MATRIX, NUMBER as arguments.
```

---

unaryOp

```
// Test for unary operations checks
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//unary operations works. it checks
// AppendLR, AppendUD ROW, COL, Addrow, AddCol
//Deleterow, deleteCOl, and gauss.it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B,C, D, E, s, t;

A= [1, 0; 0, 1;];
B= [0, 1; 0 , 2;];
C= [1, 1; 1, 1;];
s = [1, 2;1, 2;];
t = [2, 4;2, 4;3, 3;];

//didn't catch the type mismatch errors
start()
        Number myEl;

        A = appendLR (A, 5);
        B = appendUD (A, 6);

        D = getRow(A, B);
        D = getCol(A);
        D= getRow(A, 10);
        D = getCol(B, 10);

        E = gauss(A);
        E = gauss(A, s);
        E = gauss(A, t);

        A = deleteRow(A, B);
        B = deleteCol(B);
        A = addRow(A, 5);
        B = addCol(B, 1);



endstart
```

---

Results for unary operations class:

```
line 29:21: expecting COMMA, found ')'
line 33:20: expecting COMMA, found ')'
```

```
line 38:24: expecting COMMA, found ')'
<AST>: unexpected end of subtree
<AST>: unexpected end of subtree
<AST>: unexpected end of subtree
BroomProgram did not compile; the following errors were detected:
* Error in start() : appendLR requires MATRIX as second argument.
* Error in start() : appendUD requires MATRIX as second argument.
* Error in start() : row requires NUMBER as second argument.
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.
* Error in start() : deleteRow col requires NUMBER as second argument.
* Error in start() : right side of an assignment is invalid.
        No value detected.  Make sure functions return value and are operational.
* Error in start() : addRow requires MATRIX as second argument.
* Error in start() : addCol requires MATRIX as second argument.
```

---

wrongNumColls

```
// Test for number of columns errors
// Author Brian Pellegrini

// This is supposed to make sure that error checking for
//assignment of matrix values works...it is supposed to fail
// MUA HAHAHHAHAHHAH

global Matrix A,B;

A= [1, 0, 1; 0, 1;];
B= [0, 1; 1, 0 , 2;];


start()
        print ("If this printed checker didn't work");
endstart
```

---

Results for test of wrong number of columns test:

```
BroomProgram did not compile; the following errors were detected:
* Error in *global declaration* : invalid matrix specification.
         Expected a row with 3 elements, but got 2
* Error in *global declaration* : invalid matrix specification.
         Expected a row with 2 elements, but got 3
```

---

Other error built-in error checking not listed is missing return types.