LAME: Linear Algebra Made Easy

David Golub
Carmine Elvezio
Ariel Deitcher
Muhammad Ali Akbar

November 2, 2010

1 Definition of a Program

A program is a sequence of variable declarations and statements.

2 Variable Declarations

2.1 Data Types

The following data types represent the complete listing of primitives available to a programmer of LAME.

- **Scalar:** This is the equivalent of a double precision floating point number in most languages. The value is 64 bit and signed. The range is as given by the IEEE floating point standard.
- **Matrix:** Matrices in LAME are represented as dynamically sized 2-dimensional arrays.
  - Elements of the matrices can be composed of either scalar values or of other matrices.
  - Matrix size is denoted using the following notation:
    - *identifier* `[`*m*`, `*n*`]`
    - The m value corresponds to the height of the matrix and the n value corresponds to the width.
  - Random access of matrix elements is handled using the following notation:
    - *identifier* `[`*x*`, `*y*`]`
    - The x value represents the row and y value represents the column.

  - Each dimension of the array uses zero-based indexing and indices must be positive.
  - Matrix literals are created by enclosing rows in curly braces.
    * Elements are separated by commas, whereas rows are separated by a semicolon. There is no semicolon needed for the final element of the matrix.
    * Every row of the matrix literal must contain the same number of elements (corresponding to the width of the matrix).
    * Whitespace is not considered in the usage of matrix literals.
  - Vectors are represented in LAME by creating a single dimensional matrix (a column vector using dimensions $n \times 1$).
  - It is possible to determine the size of a dimension of a matrix using the `size_`*`dimension`* keyword.
    * There are two variants of the keyword:
      · `size_rows` *`matrixname`*, returns the number of rows of matrixname
      · `size_cols` *`matrixname`*, returns the number of columns of matrixname

- **Boolean:** Accepts values of true and false which can be used with logical operators. Scalars cannot be implicitly converted to booleans.
- **String:** String values are collections of ASCII characters enclosed in double quotes.
  - Literals can be concatenated to other literals.
  - When attempting to print values of other data types, implicit conversion to string type occurs.

2.2 Identifiers

- Identifiers must begin with a letter (uppercase or lowercase), and may contain letters, digits, and underscores.

2.3 Scope

- Variables have global scope.
- Variables declared can be reassigned but their memory allocation will remain throughout the duration of the program.

2.4 Variable Declaration

Declaration of variables follow this general format:

- Without initialization
  - *datatype identifier*;
- With initialization
  - *datatype identifier = expression*;
- Type-specific initialization format:
  - **Scalar:**
    - `scalar name = 9.8165;`
  - **Matrix:**
    - `matrix name = {1, 2, 3; 4, 5, 6; 7, 8, 9};`
      - Setting initial values is optional. If not initialized, the matrix is created as a zero matrix of size 1×1.
    - This corresponds to a 3×3 matrix.
    - Row elements are separated by a comma, whereas column elements are separated by a semicolon.
  - **Boolean:**
    - `boolean name = true;`
    - Initial values must be either true or false.
  - **String:**
    - `string name = "hello";`

3 Statements

3.1 Assignment Statements

Assignment statements in LAME are of the form

lvalue = *expression*;

The expression is evaluated and the lvalue is set to equal its value. The lvalue may be either a variable or a matrix/vector element access.

Example:

```
y = (x + 5) * 7;
A[1, 5] = 8;
```

3.2 Redimensioning of Matrices

The `dim` statement allows for the redimensioning of matrices. It is of the form

`dim` *`matrixname`*`[`*`rows`*`,` *`cols`*`];`

where both *`rows`* and *`cols`* must be integers greater than 0, or an exception is raised.

In the event that either *`rows`* or *`cols`* is **less** than the current respective sizes of the rows and/or columns, the matrix is truncated accordingly. In the event that either *`rows`* or *`cols`* is **greater** than the current respective sizes of the rows and/or columns, the appropriate number of rows and/or columns are appended to the matrix, with each newly appended entry initialized to 0.

Example:

```
dim A[3, 2];
```

3.3 If Statements

The `if` statement is of the form

`if(`*`condition`*`)` *`statement`*

where the statement following the `if` is only executed if the condition evaluates to true. Otherwise execution continues following the statement block.

The `else` statement, which cannot be used unless it immediately follows an `if` statement section, is of the form

`else` *`statement`*

where the statement block following the `else` is only executed if the preceding `if` evaluates to false.

Example:

```
if(A < B)
    print A;
else
    print B;
```

3.4 While Loops

The `while` statement is an iterative loop of the form

`while(`*`condition`*`)` *`statement`*

where the expression must evaluate to a boolean true or false. If the expression is true, the statement is executed. The expression is then re-evaluted, and the statement is executed so long as the condition is true.

Example:

```
while(A[1] < B[1])
    A[1] = A[1] + 1;
```

3.5 Block Statements

A block statements is a list of statements enclosed in curly braces. It is generally used to create an if statement or while loop with multiple statements that are executed conditionally or repeatedly.

Example:

```
if(x >= 5) {
    y = x + 7;
    z = 3 * y;
}
```

3.6 Basic I/O

Printing to standard out is done using the `print` keyword, followed by the expression to print.

Example:

```
print A;
```

4 Expressions

4.1 Literals

There are four types of literals in LAME: scalar literals, matrix literals, string literals, and Boolean literals. A scalar literal takes the form of a nonempty sequence of digits, optionally followed by a decimal point and a second nonempty sequence of digits, optionally followed by the letter E in either capital or lowercase and a nonempty sequence of digits indicating an exponent. If it is present, the sequence of digits indicating the exponent may be preceded by a minus sign to indicate that the exponent is negative. The following are examples of scalar literals:

```
5
7.3
7e-5
1.1e7
```

Matrix literals are composed of a sequence of rows separated by semicolons and enclosed in curly braces. Each row consists of a sequence of scalar literals

separated by commas. All of the rows in a matrix literal must have the same number of scalar literals in them. The following are examples of matrix literals:

```
{
    1, 0, 0;
    0, 1, 0;
    0, 0, 1
}

{ 1.5, 0.5; 0.5, 1.5 }
```

A string literal is composed of a quotation mark, followed by a possibly empty sequence of string characters and/or escape sequences, followed by another quotation mark. A string character is any character except a quotation mark, backslash, or newline. An escape sequence is a backslash followed by either a quotation mark, a backslash, the lowercase letter N, or the lowercase letter T. These escape sequences denote the presence of a quotation mark, backslash, newline, or tab, respectively, in the string. The following are examples of string literals:

```
"Hello, World!"
"Say, \"Hello!\""
"First line\nSecond line"
"C:\\FOLDER\\FILENAME.EXT"
```

There are only two possible Boolean literal values, which are denoted by the keywords `true` and `false`.

4.2 Unary Arithmetic Operators

Two unary arithmetic operators are provided: negation and transposition. Negation is a prefix operator indicated by a minus sign and has the standard mathematical meaning. It is valid on scalars and matrices. Attempting to negate a string value will yield a compiler error. Transposition is a postfix operator indicated by an apostrophe. It is valid on matrices and scalars. On matrices, it has the standard mathematical meaning. On scalars, it has no effect. Transposing a scalar yields that same scalar. Attempting to transpose a string will yield a compiler error.

4.3 Binary Arithmetic Operators

LAME provides five binary arithmetic operators: addition, subtraction, multiplication, division, and exponentiation. These operators are denoted by the plus sign, minus sign, asterisk, slash, and caret, respectively. All five operators are infix operators and can be applied to scalars and have the standard mathematical meaning. Addition, subtraction, and multiplication can be applied

to pairs of matrices and have the standard mathematical meaning. A runtime check will be performed to ensure that the two matrices are of sizes such that the operations can actually be performed. Multiplication can be applied to a matrix and a scalar and has the standard mathematical meaning. Division can be applied to a matrix and a scalar and has the effect of multiplying the matrix by the reciprocal of the scalar. The matrix must be the first operand and the scalar must be the second. Exponentiation can be applied to a matrix and a scalar and has the effect of multiplying the matrix by itself a number of times given by the scalar. The matrix must be the first operand and the scalar must be the second. A runtime check will be performed to ensure that the matrix is square. The addition operator can be applied to strings and in this context represents string concatenation. All other combinations of operand data types will yield a compiler error.

4.4 Relational Operators

LAME provides six relational operators: less than, greater than, less than or equal to, greater than or equal to, equal to, and not equal to. These operators are denoted by `<`, `>`, `<=`, `>=`, `==`, and `!=`, respectively. The last two of these operators are valid on all data types. The first four are valid on scalars and strings only. Attempting to use them on matrices or Boolean values will yield a compiler error. All six relational operators produce a Boolean value indicating whether the corresponding relation holds.

4.5 Logical Operators

LAME provides three logical operators: logical and, logical or, and logical not. These operators are denoted by `&&`, `||`, and `!`, respectively. The first two are binary operators that take a pair of Boolean values and produce a Boolean value giving the result of the logical operation. Logical not is a unary prefix operator that takes a single Boolean value and produces a Boolean value giving its logical negation. All three logical operators will yield a compiler error if they are used on scalars, matrices, or strings.

4.6 Operator Precedence

The operator precedence and associativity for LAME is as given in the following table. Operators of the highest precedence are at the top of the table.

Operators	Associativity
Transposition	Not applicable
Negation	Not applicable
Logical not	Not applicable
Exponentiation	Left-associative
Multiplication, division	Left-associative
Addition, subtraction	Left-associative
Less than, greater than, less than or equal to, greater than or equal to	Left-associative
Equal to, not equal to	Non-associative
Logical and	Left-associative
Logical or	Left-associative

4.7 Matrix/Vector Element Access

This section describes the syntax of the matrix element access and the vector element access.

4.7.1 Matrix Element Access

The syntax of a matrix element access is as follows:

```
A[i,j]
```

This represents the matrix element A_{ij} where A is the name of the matrix, i is the row number and j is the column number.

The behavior of the matrix element access is as follows:

- For the first row, $i = 0$; for second row, $i = 1$; and so on. Similarly, for first column, $j = 0$; for second column, $j = 1$; and so on. Formally, the element access requires two scalars from set of natural numbers (including zero) i.e. $i, j \in N_0 = \{0, 1, 2, ...\}$.
- The matrix element access `A[i,j]` can be an lvalue. For example,

  ```
  A[i,j] = 2;
  ```

  assigns numeric value 2 to the matrix element A_{ij};
- The matrix element access `A[i,j]` returns the numeric value of the matrix element A_{ij}. For example,

  ```
  B[i,j] = A[i,j] + 2;
  ```

  adds 2 to the numeric value of A_{ij} and assigns it to the matrix element B_{ij}.

4.7.2 Vector Element Access

There is no vector type in LAME. However, the programmer is allowed to perform vector element access on a matrix. The syntax of a vector element access is as follows:

```
V[i]
```

This represents the element $V_{i,0}$ where V is the name of the matrix, i is the row number. The behavior of the Vector element access is as follows:

- For first element, $i = 0$; for second element, $i = 1$; and so on. Formally, the vector element access requires a scalar literal from set of natural numbers (including zero), i.e. $i \in N_0 = \{0, 1, 2, ...\}$.
- The vector element access `V[i]` can be an lvalue. For example,

  ```
  V[i] = 2;
  ```

  assigns numeric value 2 to the element $V_{i,0}$;
- The vector element access `V[i]` returns the numeric value of the element $V_{i,0}$. For example,

  ```
  X[i] = V[i] + 2;
  ```

  adds 2 to the numeric value of element $V_{i,0}$ and assigns it to the vector element $X_{i,0}$;

4.8 Implicit Casting

In `LAME`, implicit casting is supported for the print statement and the string concatenation operator. Although the print command is for the string data type, the user can call the print command on scalars and matrices. Similarly, string concatenation operator (+) when applied to 'a string and a scalar' or 'a string and a matrix' results in implicit casting of the scalar or matrix to string literal before actual concatenation is done. Implicit conversion to string data type takes place as specified below:

4.8.1 Scalar Literal to String Literal

A scalar literal is converted to a string literal in the standard sense. As an example, consider the statements

```
print 5;
x = 102;
print x;
print "Value of x is " + x;
```

The constant scalar value `5` is converted to string literal `"5"`, while the third statement results in implicit casting of `102` to string `"102"`. In the fourth statement, scalar x is implicitly casted to string literal `"102"`, then concatenated with the other string literal and then passed to print command. As a result, the print keyword sees this string literal as its operand: `"Value of x is 102"`.

4.8.2 Matrix Literal to String Literal

When a matrix is passed to the print command or a string concatenation operator along-with a string literal as the other operand, it is implicitly casted to a string literal with a specified format that is specified in the rules below.

- **Matrix Elements:** The matrix elements are all scalar and follow the rules specified in the previous paragraph about implicit casting from scalar literals to string literals.
- **Rows:** When the matrix is converted to a string literal, each row is separated by a newline (\n) character.
- **Columns:** When the matrix is converted to a string literal, each row is separated by a tab (\t) character.

As an example of implicit conversion from matrix to string literal,

```
A = { 1, 0, 0; 0, 1, 0; 0, 0, 1 };
print A;
print "A =\n" + A
```

is converted to string literal

```
"1\t0\t0\n0\t1\t0\n0\t0\t1\n"
```

and printed as

```
1    0    0
0    1    0
0    0    1
```

in first print statement, and printed as

```
A =
1    0    0
0    1    0
0    0    1
```

in second print statement.

5 Keyword List

The following are keywords in LAME and therefore cannot be used as identifiers:

```
boolean
else
false
if
matrix
print
scalar
size_cols
size_rows
string
true
while
```