CMAT Language - Language Reference Manual
COMS 4115

Language Guru: Michael Berkowitz (meb2235)
Project Manager: Frank Cabada (fc2452)
System Architect: Marissa Ojeda (mgo2111)
Tester: Daniel Rojas (dhr2119)

Contents

1. Introduction **2**

2. Types **2**
2.1 Basic Data Types 2

3. Lexical Conventions **3**
3.1 Identifiers 3
3.2 Keywords 3
3.3 Constants 4
3.4 Comments 4
3.5 Operators 4
3.6 Precedence 5

4. Syntax Notation **6**
4.1 Expressions 6
4.1.1 Primary Expressions 6
4.1.2 Postfix and Prefix Expressions 6
4.1.3 Matrix References 6
4.1.4 Function Calls 6
4.2 Declarations 7
4.2.1 Type Specifiers 7
4.2.2 Matrix Declarations 7
4.2.3 Function Declarations 7
4.3 Initialization 8
4.3.1 int 8
4.3.2 double 8
4.3.3 bool 8
4.3.4 string 8
4.3.5 matrix 9
4.4 Statements 9
4.4.1 Expression Statement 9
4.4.2 Compound Statement 9
4.4.3 Selection Statement 9
4.4.4 Iteration Statement 10

5. Standard Library Functions **10**
5.1 Math 10
5.2 Vectors 11
5.3 Matrix 11
5.4 I/O 12

6. Semantics **12**

7. Examples **12**

8. References **13**

1. Introduction

Welcome to the CMAT language reference manual! Our team's shared interest in mathematics led us to decide on creating a language centered around matrix manipulation. Matrices are widely used throughout math, computer science, and physics to represent data and mathematical equations. CMAT aims to make matrix manipulations and other such linear algebra operations easier for applications. CMAT is inspired by C and MATLAB, taking the best parts of both to produce a language with high versatility.

Ideally, we want to allow easy, efficient computation and matrix operations without sacrificing the structure of a full programming language. Some other potential applications of our language include finding eigenvalues and eigenvectors, finding the inverse of a matrix, performing linear transformations on vectors, and solving numerical methods.

2. Types

2.1 Basic Data Types

Primitive Data Types

Name	Description
`int`	Integers are 4 bytes.
`bool`	True and False, 1 and 0.
`double`	64-bit floating point number
`null`	Absence of data

Supported Data Types

Name	Description
`String`	A series of unsigned bytes where each byte refers to a character in the ASCII table.
`T [int i[, int j]]`	This can declare a vector or a matrix consisting of elements of data type T. If only i is given, it is an i-dimensional vector. If j is also given, it is an i by j matrix.

3. Lexical Conventions

3.1 Identifiers

Identifiers can be created with any uppercase or lowercase letter and can then be followed by any arrangement of uppercase or lowercase letters, numbers and underscores.

3.2 Keywords

The following identifiers are reserved for the use as keywords and may not be used otherwise:

Basic Keywords

Name	Description
`for`	Iteration until condition not met
`while`	Loops until condition is not met
`if`	Dynamic if accepts 1/0/True/False/null
`else`	Paired with an ‘if’ statement
`elseif`	Paired with an if statement. Dynamic elseif accepts 1/0/True/False/null
`break`	Will break out of the closest enclosing for or while loop
`main`	First function executed in a program
`return`	Returns a value of a function
`void`	Nonexistent value. Used as a data type for functions that do not return anything.
`const`	Denotes a constant identifier
`true`	1
`false`	0

3.3 Constants

A constant is declared by using the `const` keyword. A constant follows the same convention as all identifiers, but every character is capitalized.

3.4 Comments

`/* */`	Block comments
`//`	Single line comments

3.5 Operators

Operators

Name	Description
`=`	Assignment operator
`+, - , *, /`	Arithmetic operators
`++`	Increment operator
`--`	Decrement operator
`>`	Greater than operator
`<`	Less than operator
`>=`	Greater than or equal to operator
`<=`	Less than or equal to operator
`==`	Returns 1 if values are equal, else returns 0
`!=`	Returns 0 if values are equal, else returns 1
`&&`	Logical AND operator
`||`	Logical OR operator
`!`	Logical NOT operator

Matrix Operators

Name	Description
`+, - , *`	Matrix arithmetic operations and scalar arithmetic operations
`[x:y:z]`	Initialize a 1-by-n matrix from x to z with a delimiter of y
`[x]`	Access specific element in vector
`[x,:]`	Access specific row of 2D matrix
`[:,y]`	Access specific column of 2D matrix
`[x,y]`	Access specific element in 2D matrix
`<,<=,>,>=`	If 2 matrices have the same dimensions, these operators compare element by element

3.6 Precedence

Operators	Precedence
`!`	Highest
`* , /`	
`+ , -`	
`< , > , <= , >=`	
`== , !=`	
`&&`	
`||`	
`=`	Lowest

4. Syntax Notation

4.1 Expressions

4.1.1 Primary Expressions

Primary Expressions are the most basic expressions which make up more

complex expressions. These include identifiers, constants, strings, or expressions in parentheses.

4.1.2 Postfix and Prefix Expressions

Postfix Expressions in CMAT include the following:

expression[expression]
expression(parameter-list)
expression.identifier
expression++
expression--

Prefix Expressions include:

++expression
--expression

4.1.3 Matrix References

Matrix elements are referenced through postfix expressions of the form:

expression[expression]

where the first expression is the identifier of an initialized matrix. In the case of a 1-dimensional matrix, the expression in brackets is simply an integer value. For a 2-dimensional matrix, the expression in brackets is a pair of comma separated values. The values can either be two integers or an integer and a colon (reference matrix operators in 3.4).

Additionally, a matrix can be referenced by its identifier if it is part of a proper matrix expression. In other words, it is preceded or followed by one of the matrix operators.

4.1.4 Function Calls

Function calls are postfix expressions of the form:

expression(parameter-list)

where *expression* is an existing function identifier and the optional *parameter-list* consists of comma-separated expressions that are passed as the function parameters.

4.2 Declarations

4.2.1 Type Specifiers

Type specifiers:

- void
- String
    - `String identifier;  //declaration of a string`
- Int
    - `int identifier;  //declaration of an int`
- double
    - `double identifier;  //declaration of a double`

4.2.2 Matrix Declarations

`T [int i[, int j]] name`	Define a vector/matrix by stating datatype brackets and name. Inside brackets can be one number for a vector, or two numbers for a matrix.

A matrix declaration follows the format seen above. This format can define a vector (1D array) or a matrix (2D array). In order to declare a matrix, we first state the primitive data type that will make up the vector/matrix. Then we have brackets with either one number inside or two numbers inside separated by a comma. If only one number is stated then a vector is created with a length of the number. If two numbers are stated then a matrix is created with the following size number by number matrix. The numbers in this declaration must be positive. The vector/matrix created is initialized to 0. Then, we specify the name of the vector/matrix which must start with a letter character followed by any number of characters including _.

4.2.3 Function Declarations

`T name (T arg, ...) { statements }`	Define a function by stating return type, name of function, and arguments in parenthesis. Braces are followed with statements inside.

To declare a function, we define the return type by stating the data type. Then, we specify the name which must start with a letter character followed by any number of characters including _. After, in parentheses are formal arguments, if any. Formal arguments are stated by writing the data type of the argument and the identifier. Multiple arguments are separated by commas. Then, braces are written in with statements in between the braces.

4.3 Initialization

When an object is declared, the declaration may include an initial value. This declaration has "=" with an initial value following it for the object. If a declaration does not include an initial value, the object (not a matrix) is initialized to null. If a vector/matrix is declared without an initial value, the vector/matrix is initialized as a zero vector/matrix. The following subsections explain how to initialize an object during and after a declaration.

4.3.1 int

During a declaration, a "=" with an initial integer value follows the declaration. After a declaration, the identifier used to declare the int will have a "=" with an initial integer value following the identifier.

4.3.2 double

During a declaration, a "=" with an initial integer value or floating-point number follows the declaration. After a declaration, the identifier used to declare the double will have a "=" with an initial integer value or floating-point number following the identifier.

4.3.3 bool

During a declaration, a "=" with an True or False follows the declaration. After a declaration, the identifier used to declare the bool will have a "=" with an initial True or False following the identifier.

4.3.4 string

During a declaration, a "=" with any character in the ASCII table follows the declaration. After a declaration, the identifier used to declare the string will have a "=" with any character in the ASCII table following the identifier.

4.3.5 matrix

A declaration is followed by a "=" with brackets specifying the elements in the matrix. If the matrix is one dimensional, then the elements can be stated with commas or a space separating them in a bracket. The number of elements in the initialization must equal the size of the 1D matrix (vector). If the matrix is 2D then the elements for each row are specified with commas or spaces

separating. Rows are separated by a semicolon. These elements must also be in brackets like the following:

```
matrix1 = [1 2 3 4];           //1D matrix (vector)
matrix2 = [1 2 3 4; 5 6 7 8]; //2D matrix (size 2x4)
```

4.4 Statements

4.4.1 Expression Statement

Expression statements consist of standalone expressions which are executed before continuing to the next statement. An expression statement in CMAT is of the form:

expression;

These are usually assignments or function calls. Expression statements may be empty if represented only by a semicolon.

4.4.2 Compound Statement

A compound statement (also known as a “block”) consists of several statements that can be used where a single statement is expected. For example, the body of a function definition is a compound statement. Compound statements are of the form:

{ declaration-list statement-list }

declaration-list and *statement-list* are both optional meaning that it is possible to have an empty compound statement. Variables declared within compound statements do not live outside of that “block.”

4.4.3 Selection Statement

A selection statement chooses a specific flow to follow based on whether a condition is met or not. In CMAT, selection statements include *if* and *else* statements in the following forms:

if (expression) statement
if (expression) statement else statement

expression must be of *bool* or arithmetic type so that the program can evaluate whether a condition is met or not. The *if* statement is executed when *expression* does not evaluate to *false, null,* or 0. Otherwise, the statement within *else* is

executed. An *else* cannot stand by itself and when used, it is paired with the last Encountered *else*-less *if* at the same block nesting level.

4.4.4 Iteration Statement

Iteration statements are meant for looping in the following forms:

while (expression) statement
for (expression; expression; expression) statement

For *while* loops, *statement* is executed as long as *expression* meets the same conditions required for an *if* statement to be executed.

For *for* loops, the three expressions within the parentheses specify at the very beginning the number of iterations that the loop will iterate over. The first expression initializes a value of any type. The second expression is evaluated in the same manner as an *if* condition. The *statement* will continue to be executed as long as this condition is met (similar to a *while* loop). The third expression is evaluated after the current iteration is executed in order to re-initialize the loop.

5. Standard Library Functions

5.1 Math

`double PI, double EUL`	Numerical values of π = 3.14159265… e = 2.71828182...
`double sqrt(int|double x)`	Returns the square root of the int or double x
`double nroot(int|double x, int n)`	Returns the nth root of int or double x
`double pow(int|double x, int n)`	Returns x^n as a double
`double cos(double x)`	Returns cosine of double x (in radians) as a double
`double sin(double x)`	Returns sine of double x (in radians) as a double

5.2 Vectors

A vector, in CMAT, is simply a 1-dimensional matrix. Vectors are treated as row vectors.

`int size(T [i] x)`	Returns the dimension of vector x as an int
`double norm(T [i] x)`	Returns the normal Euclidean length of vector x (square root of the sum of the components squared) as a double
`double dot(T [i] x, T [i] y)`	Returns the dot product of two same-dimensional vectors x and y
`T [j] roots(T [i] x)`	For a vector x representing a polynomial $f(x) = c_n x^n + c_{n-1} x^{n-1} + ... + c_1 x + c_0$ as x = [c_n, c_{n-1}, ..., c_1, c_0], return a vector of the roots of f(x)

5.3 Matrix

`T typeof(T [i,j] A)`	Returns the data type that matrix A holds
`double det(T [n,n] A)`	Returns the determinant of square matrix A
`T [n,n] identity(int n)`	Returns the n by n identity matrix
`T [j,i] transpose(T [i,j] A)`	For an i by j matrix, returns the transpose a j by i matrix
`T [n,n] inverse(T [n,n] A)`	If it exists, returns the inverse of a square matrix A Else, return null
`T [n] eigenvalues(T [n,n] A)`	Returns the eigenvalues of a square n by n matrix A in an n-dimensional vector
`T [n,n] eigenvectors(T [n,n] A)`	Returns the eigenvectors of a square n by n matrix A in an n by n matrix with each row corresponding to an

	eigenvector

5.4 I/O

`void print_line(String s)`	Prints string s to stdout and then prints a newline
`String get_line()`	Gets a '\n'-terminated String from stdin
`String itos(int i)`	Returns the int i as a String
`String dtos(double d)`	Returns the double d as a String
`String btos(bool b)`	Returns the bool b as either "true" or "false"
`double itod(int i)`	Returns the int i cast to a double (e.g. int 1 → double 1.0
`int dtoi(double d)`	Returns the double d cast to an int by truncating the fractional part (e.g. double 1.8 → int 1)

6. Semantics

In CMAT, every statement must end with a semicolon ";". Code blocks in control flow statements (if, else, elseif, for, while) must always be enclosed in braces. Braces provide more visual understanding of scope.

The program begins with a main function in a file. The main function must always have a return type int. The main function calls other functions defined which in turn may call other functions or files. When a function is called the number actuals must match the number of formal arguments in the function declaration. If a function has a return type, the end of the function must return type specified in the function declaration. If a return object from a function is being stored in a variable, the variable type must match the type of the return object from the function.

7. Examples

```
int main() {
     print_line("2D Rotation");
     int [2] x1, int [2] x2;         // Declares 2 2D vectors
     int [2,2] rot = [0, -1; 1, 0];      // Declares and initializes
a
```

```
                                        // 2x2 rotation matrix for
                                       // theta = 90°
     x1 = [1/sqrt(2),1/sqrt(2)];       // Initializes x1 to be unit
                                       // vector in direction of
                                       // theta = 45°
     x1 = 2*x1;                             // Multiply x1 by 2
     x2 = rot*x1;                      // Initialize x2 to be 2*x1
                                       // rotated by 90°. Should be
                                       // x2 = [-2/sqrt(2),2/sqrt(2)]
     return 0;
}
```

```
T [j,i] transpose(T [i,j] M) {
     typeof(M) [j,i] tr;                    // Create a j by i matrix
                                            // to hold the transpose
     int ii, jj;
     for(ii=0; ii < i; ii++) {
          for(jj=0; jj < j; jj++) {
               tr[jj, ii] = M[ii, jj];  // Set every element
                                            // tr[j,i] to M[i,j]
          }
     }
     return tr;
}
```

8. References

Kernighan, Brian W., and Dennis M. Ritchie. "Appendix A - Reference Manual." The C Programming Language. Englewood Cliffs, NJ: Prentice Hall, 1988. N. pag. Print.