All That Matrix Language Reference Manual

[Type the document subtitle]

7/2/2014
Columbia University COMS W4115
Stefanie Zhou (sz2475)

Contents

1. Introduction

All That Matrix is a programming language targeted at matrix manipulations with emphasize on the clear syntax and a lightweight compiler. All That Matrix provides intuitive matrix related operators with the goal of avoiding as many built-ins as possible and making it easy to write custom functions in the language itself. This language reference manual is inspired by the C reference manual [1].

2. Lexical Elements

2.1 Comments

Comments are delineated with an opening /* and closing */. The compiler will ignore comments. Nesting of comments is not allowed.

```
/* This is a comment */
```

2.2 Identifiers

Identifiers are sequences of characters that must start with a lower case letter and can be followed by any number of upper-case letter, lower-case letters, digits, and underscores, used for naming variables and functions. Identifiers are case sensitive, so “foo” and “Foo” are distinct.

```
Identifier -> [a-z][a-zA-Z_0-9]*
```

2.3 Keywords

Keywords are reserved for use as part of the programming language and therefore, cannot be used for any other purposes.

int	float	boolean	char
cvector	rvector	matrix	fun
if	else	for	return
true	false	import	export
print			

2.4 Constants

There are a total of five constants in All That Matrix: numeric, string literal, boolean, vector, and matrix.

2.4.1 Numeric Constants

A numeric constant can be either an integer constant or a float constant. An integer constant is a sequence of digits. A float constant is composed of an integer part, a decimal point, and a trailing character ‘f’.

Numeric Constant -> Integer Constant | Float Constant

Integer Constant -> [0-9]+

Float Constant -> [0-9]+.[0-9]*f

2.4.2 String Literal Constants

String literal constants are delineated by double quotation marks and can contain any character.

String Literal Constant -> “[any character]”

2.4.3 Boolean Constants

Boolean constant can either be true or false

Boolean -> true | false

2.4.4 Vector Constants

The two types of vector constants are row vector and column vector. Vector constants are enclosed within square brackets with whitespaces separating the elements, and a trailing ‘r’ for row vector or a trailing ‘c’ for column vector.

[1 2 3]r is a row vector

[4 5 6]c is a column vector

2.4.5 Matrix Constants

Matrix constant are enclosed in square brackets with vertical bars separating the rows and whitespaces separating the columns. Matrix constants can be filled with vector or numeric constants.

[1 2 3 | 4 5 6] is a matrix defined by numeric constants

[c1 c2] is a matrix defined by two column vectors

[r1 | r2 | r3] is a matrix defined by three row vectors

3. Types

3.1 Atomic Types

Atomic types are basic types that are used to build compound types. The four supported atomic types are

int float

Boolean char

3.2 Compound Types

The three supported compound types are

rvector

cvector

matrix

4. Expressions and Operators

4.1 Expressions

An expression consists of at least one operand and zero or more operators. Operands are one of the typed objects such as matrix and can be an identifier, a constant, or a function call that returns a value.

4.2 Unary Operators

x’ transposes a matrix

x| returns the determinant of the matrix x

x! returns the inverse of the matrix x

4.3 Binary Operators

These are the binary operators that follow the standard arithmetic and matrix operation rules. These operators are valid between two objects of the same type for integers and floats, and the result is of the same type. However, for vectors and matrices, the types between the two expressions can differ.

For example, multiplication between an integer and a matrix is equivalent to scaling the matrix by the integer, whereas multiplication between two matrices follows the standard matrix multiplication rules.

In other words, the behavior of the operators depends on the type of the operands provided. For example, when adding two integers: 5 +10, the result is 15. When adding two row vectors [x1 x2]r + [y1 y2]r, the result is the row vector [x1+y1 x2+y2]r.

expression + expression

expression - expression

expression * expression

expression / expression

4.4 Logical Operators

Logical operators between two boolean expressions can be used in control flow.

expression && expression

expression || expression

expression == expression

4.5 I/O Expressions

All That Matrix can read in inputs and produce outputs with the I/O expressions defined within the language.

identifier = import(filename)

export(filename, identifer)

4.6 Other Functions

All That Matrix also provides a limited set of built-in functions for type matrix to retrieve information about the object such as col_count(matrix) and row_count(matrix).

5. Statements

All statements must end with a semi-colon. All statements either declare a variable or modify an existing variable. If and for statements are supported for flow control and curly brackets can group statements together. The rules are the same as the C language.

6. Declarations

6.1 Program Definition

A program in All That Matrix consists of a sequence of statements, which are executed in order.

6.2 Function Declarations

A function declaration must start with the keyword fun, followed by the name of the function, and a list of zero or more parameters separated by commas and enclosed in parenthesis. Functions in All That Matrix must be declared and implemented simultaneously. The return type for functions are not explicitly declared.

Here is an example of a function declaration with two parameters.

```
fun foo (matrix a, matrix b) {

    return a+b;

}
```

7. Scope

A declared object is only visible in the scope enclosed by the nearest curly bracket pair. Declarations made within functions are visible only within those functions. A declaration is not visible to declarations that came before it. An identifier declared outside of any curly bracket pairs is a global variable, and thus, is accessible from anywhere of the program.

8. Example

```
/* A example program in All That Matrix that divides two matrices without using the binary
division operator */

r1 = [1 2]r;
r2 = [2 1]r;
matrix_a = [r1 r2];
matrix_b = import(myMatrix.txt);

a_cols = col_count(matrix_a);
b_rows = row_count(matrix_b);

if (a_cols == b_rows){
    b_inverse = b!;
    result = a * b;
    export(output.txt, result);
}
```

References

[1] B. W. Kernighan and D. Ritchie. The C Programming Language, Second Edition. Precentice-Hall, 1988.