# VSCOde Language Reference Manual

Jessica Cheng (jc4687)
Anna Lu (ajl2256)
Hana Mizuta (hm2694)
Spencer Yen (ssy2121)
Kenny Yuan (kky2114)

October 15, 2018

## Contents

## 1 Introduction and Motivation

VSCOde is a language designed to analyze and manipulate images, inspired by the high definition mapping technology used by autonomous vehicles.

VSCOde performs various image image rotation, color adjustments (saturation, brightness, contrast, etc.), basic edge detection, and custom filters. Images are represented as matrices, granting users greater control over the individual pixels in an image and allowing easy application of filters through matrix transformations.

VSCOde syntax draws from C, Matlab, and Python with its matrix-focused nature.

## 2 Types

### 2.1 Primitives

| **Type** | **Description** |
|---|---|
| `char` | 8-bits (smallest addressable unit) |
| `int` | 32-bit signed integer |
| `double` | 64-bit float point number |
| `bool` | 8-bit boolean variable |
| `string` | Array of ASCII characters |

### 2.2 Structures

| **Structure** | **Description** | **Syntax** |
|---|---|---|
| `matrix` | Mutable data structure storing a collection of primitive data types with set dimensions (n, m) that are immutable. All elements of a matrix must be of the same type.<br>Size = total size of all of the elements in the matrix Saved in memory with pointer to the first element. Every other element is placed next to the preceding element. | `[`<br>`  -1, -1, -1;`<br>`  -1, 8, -1;`<br>`  -1, -1, -1`<br>`]` |

## 3 Lexical Conventions

### 3.1 Identifiers

The convention for identifiers is camel case, where the first letter of the identifier is lowercase, but every subsequent first character of a word is uppercase (e.g. sampleImage)

### 3.2 Keywords

The keywords listed in the table below are reserved by the language, and cannot be used as identifiers.

| Keyword | Description |
|---|---|
| `func` | Function declaration. Follows syntax `func name(type varName) -> returnType` |
| `->` | Denotes return type of function |
| `return` | Ends the current function execution and returns a value |
| `void` | Indicates that function has no return value |
| `true` | Boolean keyword for true |
| `false` | Boolean keyword for false |
| `int` | 32-bit signed integer |
| `double` | 64-bit floating point number |
| `bool` | 8-bit boolean |
| `string` | Array of ASCII characters |
| `matrix` | Mutable data structure storing any size of primitive data types |
| `for` | Standard for loop that executes statements when a condition related to a variable that gets incremented or decremented is true. Var must be initialized before being used in the loop. Follows syntax `for(var; condition; incr) { statements }` |
| `while` | Standard while loop that executes statements when a condition holds true. Follows syntax `while(condition) { statements }` |
| `if` | Standard if, else condition clause. Follows syntax `if (condition) { statements } else { statements }` |
| `else` | |
| `continue` | Stops the current iteration of a for or while loop, and starts the next iteration |
| `break` | Stops the iteration of the immediate enclosing for or while loop |

## 3.3 Literals

### 3.3.1 Integer Literals

A sequence of one or more numerical digits representing an integer. Example: `[0-9]+`

### 3.3.2 Double Literals

A sequence of zero or more numerical digits followed by a `‘.’`, followed by one or more numerical digits. Example: `[0-9]*‘.’[0-9]+`

### 3.3.3 Character Literals

A single character enclosed by a pair of single quotation marks representing an unnamed char. Example: `’a’`

### 3.3.4 Boolean Literals

Consists of two keywords `true` or `false`

### 3.3.5 String Literals

A sequence of character primitives enclosed by a pair of double quotation marks representing an unnamed string. Example: `"this is a string"`

### 3.3.6 Tuple Literals

A comma separated sequence of data types enclosed by parentheses. You can return a tuple, but cannot move it around as a type (for the scope of our language, it is not necessary).
Example: `(int, matrix, boolean)`

### 3.3.7 Matrix Literals

A sequence of primitive data types enclosed by square brackets, with rows delimited by semicolons, and items delimited by commas.
Example: `[-1,-1,-1; -1,8,-1; -1,-1,-1] =`

```
[
    -1, -1, -1;
    -1, 8, -1;
    -1, -1, -1
]
```

## 3.4 Comments

### 3.4.1 Single-line Comments

Single-line comments are denoted by `//`. Example:

```
// this is a single-line comment
```

### 3.4.2 Multi-line Comments

Multi-line comments are denoted by `/* */`. VSCOde does not support nested multi-line comments. Example:

```
/* this is a
   multi
   line
   comment */
```

## 3.5 Operators

### 3.5.1 Arithmetic Operators

| Operator | Description |
|---|---|
| + | Addition (binary operator between two ints or two doubles). Will throw error if called on an int and double. |
| - | Subtraction (binary operator between two ints or two doubles). Will throw error if called on an int and double. |
| * | Multiplication (binary operator between two ints or two doubles). Will throw error if called on an int and double. |
| / | Division (binary operator between two ints or two doubles). Will throw error if called on an int and double. |
| % | Modulo (binary operator between two ints or two doubles). Will throw error if called on an int and double. |
| <, >, <=, >= | Greater than, less than, greater than or equal to, less than or equal to, equal to, not equal to. Binary operator between two ints or two doubles. Will throw error if called on an int and double. |

## 3.6 Matrix Operators

| Operator | Description | Example |
|---|---|---|
| + | Addition (binary operator between two matrices or a scalar and matrix). Will throw error if matrix dimensions are incompatible. | `3 + [-1, -1, -1; -1,  8, -1]`<br>`= [2, 2, 2; 2, 11, 2]`<br><br>`[1, 1] + [2, 2]`<br>`= [3, 3]` |
| - | Subtraction (binary operator between two matrices or a scalar and matrix). Will throw error if matrix dimensions are incompatible. | `[-1, -1, -1; -1,  8, -1] - 3`<br>`= [-4, -4, -4; -4, 5, -4]`<br><br>`[1, 1] - [2, 2]`<br>`= [-1, -1]` |
| * | Multiplication (binary operator between two matrices or a scalar and matrix). Will throw error if matrix dimensions are incompatible. | `3 * [-1, -1, -1; -1,  8, -1]`<br>`= [-3, -3, -3; -3,  24, -3]`<br><br>`[1, 2, 3; 2, 3, 4] * [1, 2; 3, 4; 5, 6]`<br>`= [22, 28; 31, 40]` |
| [r, c] | Matrix access. | `int matrix [3,3] M =`<br>`    [`<br>`        -1, -1, -1;`<br>`        -1,  8, -1;`<br>`        -1, -1, -1`<br>`    ];`<br>`print(M[1, 2]) // should print element in`<br>`    2nd row, 3rd column: -1` |

# 4 Syntax Notations

## 4.1 Expressions

### 4.1.1 Precedence and Associativity Rules

| Precedence | Operator | Token | Associativity |
|---|---|---|---|
| 1 | Parenthetical Grouping | () | Left |
| 2 | Array/Matrix Subscript<br>Function Call | []<br>() -> | Left |
| 3 | Unary Operator | ! | Right |
| 4 | Binary Multiplicative Operators | * / % | Left |
| 5 | Binary Additive Operators | + - | Left |
| 6 | Comparative Operators | < <= => > | Left |
| 7 | Equality | == != | Left |
| 8 | Assignment | = | Left |
| 9 | Sequencing | ; | Left |

### 4.1.2 Type Conversions

The user must explicitly cast between `doubles` and `ints`, and strings with any other type to generate the string notation. (ie. `string x = (string) 5;`)

### 4.1.3 Equality

The equality operators (`==` and `!=`) are structural for primitive types, meaning that they recursively compare the values of primitives. However, all other types use referential equality.

### 4.1.4 Subscripts

A postfix expression in square brackets is a subscript, whose expression has type matrix.

## 4.2 Declaration

### 4.2.1 Matrix Declaration

```
type matrix [m,n] name = [a, b, c; e, f, g; h, i, j];
```

Example:

```
int matrix [3,3] mat = [1, 2, 3; 4, 5, 6; 7, 8, 9];
```

The matrix specifier, followed by the immutable dimensions of the matrix in brackets, define the variable as a matrix type of fixed dimensions. The elements within the matrix are of any primitive type. Semicolons separate every row, while commas separate elements within each row.

### 4.2.2 Function Declaration

```
func name(type varName) -> returnType
```

Use the keyword `func` to declare this is a function declaration. Then follows the user defined function name, followed by parentheses. In the parentheses, the arguments (type and name of argument) that the function accepts is defined. After the parentheses, an arrow represented by `->` followed by a data type signifies the return type of the function

# 5 Standard Library Functions

## 5.1 Functions

| **Name** | **Description** | **Return Type** |
|---|---|---|
| print(string s) | Prints argument to standard output | void |
| dim(matrix m) | Gets the dimensions of an object | (int, int) |
| load(string name) | Loads an image into a 3-tuple of red, green, blue matrices | (matrix, matrix, matrix) |
| save(matrix r, matrix g, matrix b, string s) | Saves 3 matrices corresponding to RGB values as a jpg image with name s | bool |

## 5.2 Load and Save Functions

The standard library will provide methods to load an image file (.jpg) as matrices and to save matrices as an image file (.jpg). Our language's representation of an image will consist of three separate matrices per image to represent the red, green, and blue color channels. These methods will be written in C to leverage OpenCV and linked to our implementation. Specifically, our load method will use OpenCV's imread() function to read an image file into a C Mat object, a multi-channeled matrix. We can then iterate through the Mat object and parse it to create three separate RGB matrices in our language's format. Our save function will work similar, leveraging OpenCV's imwrite() function to parse our matrices into a Mat object and write it back into an image file.

### 5.3 Matrix Transformations

| **Method** | **Description** |
|---|---|
| transpose(matrix m) | Transposes matrix m |
| replace(matrix m, int a, int b) | Replaces every instance of a in matrix m with b |
| multiply(matrix a, matrix b) | Multiplies matrix a by matrix b |
| rotate(matrix a, double deg) | Rotates matrix a by deg degrees by multiplying the matrix by [[cos(deg), sin(deg)], [-sin(deg), cos(deg)]] |
| convolute(matrix a, matrix b) | Convolutes matrix a with matrix b |

## 6 Code Samples

### 6.1 GCD Algorithm

```
// greatest common denominator function in VSCOde
func gcd (int m, int n) -> int {
    while (m > 0) {
        int c = n % m;
        n = m;
        m = c;
    }
    return n;
}
```

### 6.2 Grayscale

```
func applyGrayscale (string imageName) -> void {

    // read image into a matrix
    // 4092 x 4092 because load matrix is of unknown size
    double matrix [4092,4092] r, g, b = load(imageName); // load returns a tuple literal

    // weighted method of grayscale
    multiply(r, 0.3);
    multiply(g, 0.59);
    multiply(b, 0.11);

    save(r, g, b, "new.jpg");
}
```

### 6.3 Edge Detection

```
func applyAllDirectionEdgeDetection (string imageName) -> void {

    // read image into a matrix
    // assume that load matrices are 4096 by 4096
    double matrix [4096,4096] r, g, b = load(imageName); // load returns a tuple literal

    // all direction edge direction matrix
    int matrix [3,3] edgeDetection =
    [
      -1, -1, -1;
      -1,  8, -1;
      -1, -1, -1
    ];

    convolute(r, edgeDetection);
    convolute(g, edgeDetection);
    convolute(b, edgeDetection);

    save(r, g, b, "new.jpg");
}
```

## 7 Team Roles

Project Manager — Jessica Cheng
System Architect — Hana Mizuta
System Architect — Kenny Yuan
Language Guru — Spencer Yen
Tester — Anna Lu

## 8 References

FaceLab Report.
Lane Detection for Self-Driving Cars with OpenCV.
Lode's Computer Graphics Tutorial.