Facelab Language Reference Manual

Xin Chen (xc2409)
Kejia Chen (kc3136)
Tongfei Guo (tg2616)
Weiman Sun (ws2517)

October 16, 2017

Contents

1 Introduction

Facelab aims to perform face detection, face recognition, filter applying and photo sticker adding among other features which enable the target users to manipulate their portrait photos with ease and accuracy. The basic syntax of this language largely resembles that of C++, excluding some of the irrelevant and hard-to-implement details such as inheritance, template, etc. With the inclusion of the matrix data type that is common to many scientific programming languages, it not only facilitates image processing related computation, but also grants users the ability to manipulate photo on a pixel scale and allows users the freedom to define and tailor their own filter to individuals preference. Moreover, by having OpenCV linked at the compiling stage, it provides access to some of the state-of-art face detection and face recognition algorithms without having to install the whole libraries of OpenCV, learning it and its companying auxiliary libraries(such as Eigen library) functionalities. A combination of these afore-mentioned features could considerably simplify real-life tasks such as adjusting photo brightness and contrast, batch-editing photos, auto-applying facial pixelization, and so on.

2 Types

2.1 Basic data types

Table 1: basic data types

type name	description
int	32-bit signed integer
double	64-bit float-point number
bool	8-bit boolean variable
string	array of ASCII characters
matrix	data structure storing bool/ints/doubles of arbitrary size

2.2 Advanced data type

Table 2: advanced data type

type name	description
image	data structure storing 3 matrices

3 Lexical Conventions

3.1 Identifiers

Identifiers consists of one of more characters where the leading character is a uppercase or lowercase letter followed by a sequence uppercase/lowercase letters, digits and possibly underscores. Identifiers are primarily used variable declaration.

3.2 keywords

The keywords listed below are reversed by the language and therefore will not be able to be used for any other purposes (e.g. identifiers)

Table 3: keywords

type name	description
for	typical for loop follows the syntax *for(init; cond; incr) stat;*
while	typical while loop follows the syntax *while(cond) stat;*
if	typical if-elseif-else condition clause follows the syntax
elseif	*if(cond) stat; elseif(cond) stat; else stat;*
else	
break	ending the iteration of the nearest enclosing for/while loop
continue	ending current iteration of a for/while loop and starting with the next iteration
return	ending current function execution and return a value or multiples values
func	signal word for function definition follow the syntax *func name(type var, ...) stat;*
true	boolean type constant
false	boolean type constant
int	32-bit signed integer
double	64-bit float-point number
bool	8-bit boolean variable
string	array of ASCII characters
matrix	data structure storing bool/ints/doubles of arbitrary size
image	data structure storing 3 matrices

3.3 Literals

3.3.1 integer literals

A sequence of one or more digits representing an un-named(not associated with any identifier) integer, with the leading digit being non-zero (i.e. [1-9][0-9]*)

3.3.2 double literals

A sequence of digits seperated by a '.' representing an un-named float-point number (i.e. [0-9]*.[0-9]*)

3.3.3 matrix literals

A sequence of digits enclosed by a pair of square brackets, and delimited by commas and semi-colons, representing an un-named 2-D matrix.

e.g. [1, 2; 3, 4] = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

3.3.4 string literals

A sequence of character enclosed by a pair of double quotation marks representing an un-named string. (i.e. ^".*" $)

3.4 Comments

Table 4: comments

/* comment */	block comment where comment could contain newline
// comment	line comment without newline

3.5 Operators

3.5.1 scalar operators

Table 5: scalar operators

=	assignment operator
+, -, *, /	arithmetic operators
%	reminder operator
$!=, ==, >, >=, <, <=$	relational operators
$\|\|$, &&, !	logical operators(OR, AND, NOT)

3.5.2 matrix operators

Table 6: matrix operators

=	assignment operator
+, -, *, /	arithmeitc operators for matrix
.*	matrix dot product
$M[i, j]$	subscript operator
$M[:, j]$	subscript j-th column
$M[i, :]$	subscript i-th row
\$	pre-defined operator whose syntax follows $image \$ filter$, which applies filter to image

3.6 punctuator

Semicolons at the end of each statement perform no operation but signal the end of a statement. Statements must be separated by semicolons.

4 Syntax Notations

4.1 Expressions

4.1.1 Precedence and Associativity Rules

Table 7: Operator Precedence and Associativity

Tokens (From High to Low Priority)	Associativity
() []	L-R
!	R-L
$	L-R
* / %	L-R
+ -	L-R
< <= > >=	L-R
== !=	L-R
&	L-R
|	L-R

4.1.2 Primary Expressions

Identifiers, literals and parenthesized expressions are all considered as ”primary expressions”.

4.1.3 Postfix Expressions

Postfix expressions involving subscripting and function calls associate left to right. The syntax for these expressions is as follows:

postfix-expression:	primary-expression
	postfix-expression [expression]
	postfix-expression (argument-expression-list)
argument-expression-list:	argument-expression
	argument-expression-list, argument-expression

4.1.4 Subscripts

A postfix expression followed by an expression in square brackets is a subscript. Usually, the postfix expression has type matrix. For a 1-D matrix, the expression has type int. For a 2-D matrix, the expression would be two values separated by a comma, the value could be an integer or a colon.

4.1.5 Function Calls

A function call is a postfix expression followed by parentheses containing a (possibly empty) comma-separated list of expressions that are the arguments to the function.

4.2 Declarations

4.2.1 Type specifiers

int
double
bool
string
matrix
image
Each variable declaration must be preceded by a type specifier which tells what type is going to be used to store that variable.

4.2.2 Matrix declarations

```
example:
matrix name = [a,b,c;d,e,f;g,h,i]
```
The **matrix** specifier define the variable as a matrix type. In the example, a–i are of type int or double. The value is surrounded by a pair of brackets. semi-colons are to separate different rows, where in every row, elements are separated by commas.

4.2.3 Function declarations

```
example:
func funcName(T arg1, ...)  {...}
```
To define a function, use the keyword **func** to declare this is a function declaration. Following by user defined function's name. In the parentheses it defines how many arguments it can be passed in and what types are they. Therefore in the calling environment, the calling statement must match the function's definition.

5 Standard Libraries Functions

5.1 image

Table 8: Standard Libraries Functions for image

Functions	Description
func load(String image_path)	load image from a file
func save(String folder_path)	save image in the folder
func recognition(String folder_path, String image_path)	recognize the face in the image by training images from a given folder
func detection(String image_path)	detect whether a image includes faces
func drawRect(String image_path, int x, int y, int width, int height)	draw a rectangle of given size at given position
func drawText(String image_path, string text, int x, int y)	draw given string at given position
func pixelize(String image_path, int x, int y, int width, int height)	apply pixelization at given position

5.2 I/O

Table 9: Standard Libraries Functions for I/O

Functions	Description
func print(String str)	print a string that is passed into the function when calling the print function

6 Sample Code

In FaceLab, 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 make the scope of the block statement more straightforward. The program begins from top down, functions don't have return types, but can return any type of variables in the function. Every function has an argument that takes 0 or more variables, surrounded by parentheses. When calling a function, the number of variables passed into the calling function must match its arguments and corresponding types. 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.

6.1 GCD Algorithm

```
func gcd(int m, int n) {
//calculate gcd of two integer number
    while(m>0)
    {
        int c = n % m;
        n = m;
        m = c;
    }
    return n;
}
```

6.2 Apply a Filter

```
image pic1 = load(../"xxx.jpg"); // built in function to load an image
matrix sharpen =
[0,-1,0;
-1,5,-1;
0,-1,0];
//define a filter.
image pic2 = pic1 $sharpen; // apply filter to image.
save("../yyy.jpg", pic2); // built in func to save an image to file
```

6.3 Face Recognition

```
string label; // define label of the given person.
label, x, y, w, h = recognition("../folder", "../xxx.jpg");
// train images from a given folder, recognize images from target path,
// return the label of the recognized person and a rectangle around the
```

```
// face, (x, y, w, h) stand for coordinates, width and height of the recta
image pic1 = load("../xxx.jpg");
pic1 = drawRect (pic1, x, y, w, h);
// draw a rectangle around the face
pic1 = drawText (pic1, identity, x + 10, y + 10);
// draw the label around the face
save("../yyy.jpg", pic1);// built in func to save an image to file
```