Facelab: A Portrait Photos Editing Language

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

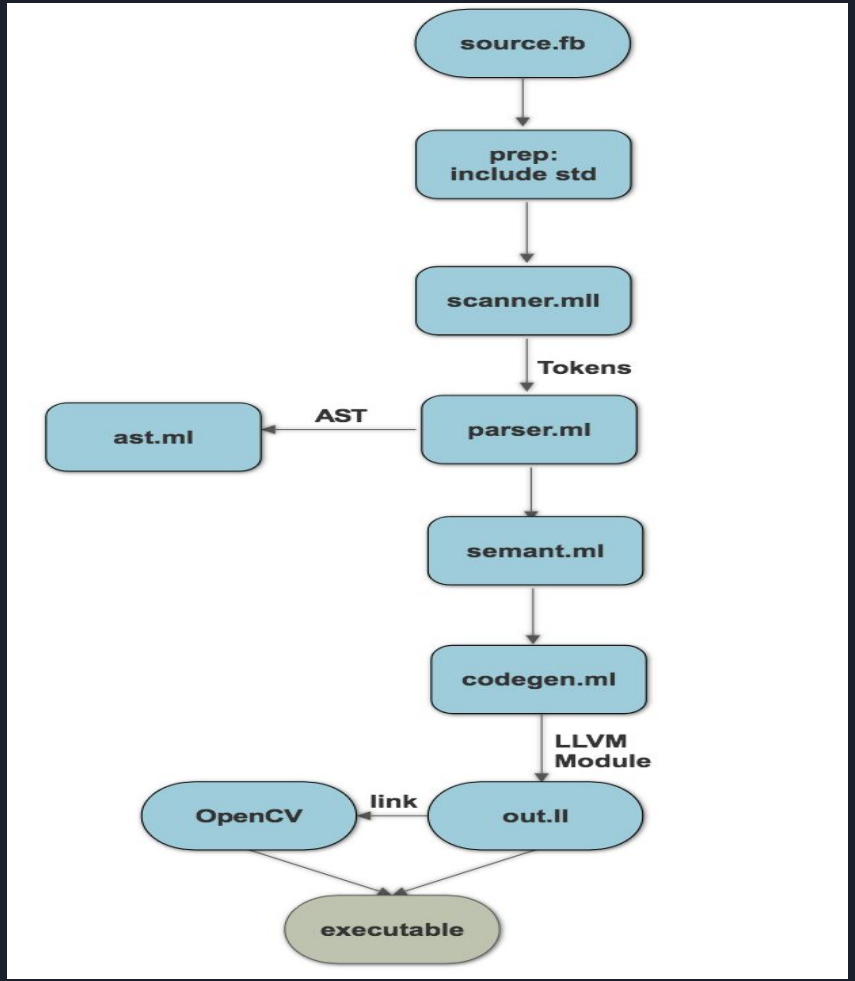
Introduction

01 face detection, filter, photo sticker

02 hybrid of C and Matlab syntax

03 have OpenCV linked

Architecture



Data types

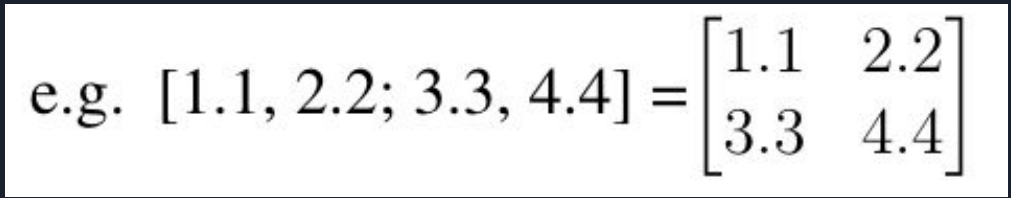
Type	Description
int	32-bit signed integer
double	64-bit float-point number
bool	1-bit boolean variable
string	array of ASCII characters
matrix	data structure storing bool/ints/doubles of arbitrary size

Keywords -- reserved by the language

- **Compound statement:** for, while, if, elseif, else.
- **Control flows:** return.
- **Boolean values:** true, false.
- **Function declaration:** func
- **Data types:** int, double, bool, string, matrix

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.



Basic Operators

`=, +, -, *, /, %`	Arithmetic operations
`!=, ==, <, >, <=, >=`	Comparison
`||, &&, !`	Logic operators
`.*`	Matrix dot product
`M[i, j]`	Access matrix entry at (i, j)
`M[i, :]`	Subscribe i-th row
`M[:, j]`	Subscribe j-th column
`$`	Operator to add filter to a matrix

Function declaration & Scoping

- function declaration can be interleaved with statements and expressions.
- both function name and variable name follow static scoping rule.

```
func f1() { return; } //void type
f1();
f2(); // error msg(Semantic error : f2 not defined.)
func f2() { return 5;} //int type
f2()
```

Return type inference

- `func f1() { return; } //void type`
- `func f2(int i, int j) { return i*j; } // int type`
- `func f3(double d) { return d+3.3; } //double type`
- `func f4(string s) { return s; } // string type`
- `func f5(int i, int j) { return i==j; } // bool type`
- `func f6(matrix m, matrix n) { return m.*n; } //matrix type`

Return type inference

- HOWTO: Search for return statement in each function, and determine the type of return expression.

- Trick for implementing recursive function call:

```
func factorial (int i)
{
    if (i != 1) { return i * factorial (i-1); }
    else { return 1; }
}
```

Return multiple values

- Since we do not provide reference operator, return multiple seems necessary.

```
func rot90(double x, double y)
{
    return -y, x;
}
double x; double y;
x, y = rot90(x, y);
```

Return multiple values

```
func rot90(double x, double y)
{
    return -y, x;
}
double x; double y;
x, y = rot90(x, y);
```

- Store all return data in a struct at llvm level, then assign to each variable one by one.

Matrix indexing

- `m[:, :]` //returns the whole matrix.
- `m[:x_high, :]` //returns row 0 to row x_high.
- `m[x_low:, :]` //returns row x_low to last row.
- `m[x_low:x_high, :]` //returns row x_low to x_high
- `m[x, y] = m[x:x, y:y]` //returns the entry at(x, y)(double)

Matrix assignment

- matrix assignment

```
m1 = m2; // their sizes do not have to agree.
```

- block assignment

```
m1[x_low:x_high, y_low:y_high] = m2; // their size must agree.
(so m[:,:] is different from m in assignment)
```

OpenCV related built-in function

- `m_r, m_g, m_b = load(path)`
  - store image in RGB order
- `save(m_r, m_g, m_b, path)`
  - save image defined by RGB matrices
- `m = face(path)`
  - use OpenCV cascade classifier
  - m is a 4 by n matrix
  - n: # of faces; row 1: x-coordinates of the center of faces; row 2: y-coordinates of the center of faces; row 3: height of the faces; row 4: width of faces.

(all above function are interface in Facelab to functions from open CV.)

Other built-in function

- size
  - i, j = size(m)
  - m: matrix; i: row; j: column
- zeros
  - m = zeros(i, j)
- double2int
  - d = double2int(i)
- int2double
  - d = int2double(i)

Std.fb

Built-in functions such as filter
is loaded at compile stage.

Filter

5*5 or 3*3 image kernels can be easily applied to rgb matrix using built in filter function $.

$ is left-associative so multiple kernels can be applied at the same time.

Matrix result = origin $ filter1 $ filter2 $...



Thank you!

Demo