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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>double</td>
<td>64-bit float-point number</td>
</tr>
<tr>
<td>bool</td>
<td>1-bit boolean variable</td>
</tr>
<tr>
<td>string</td>
<td>array of ASCII characters</td>
</tr>
<tr>
<td>matrix</td>
<td>data structure storing bool/ints/doubles of arbitrary size</td>
</tr>
</tbody>
</table>
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.

\[
\text{e.g. } [1.1, 2.2; 3.3, 4.4] = \begin{bmatrix}
1.1 & 2.2 \\
3.3 & 4.4
\end{bmatrix}
\]
## Basic Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=, +, -, *, /, %</code></td>
<td>Arithmetic operations</td>
</tr>
<tr>
<td><code>!=, ==, &lt;, &gt;, &lt;=, &gt;=</code></td>
<td>Comparison</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>.*</code></td>
<td>Matrix dot product</td>
</tr>
<tr>
<td><code>M[i, j]</code></td>
<td>Access matrix entry at (i, j)</td>
</tr>
<tr>
<td><code>M[i, :]</code></td>
<td>Subscribe i-th row</td>
</tr>
<tr>
<td><code>M[:, j]</code></td>
<td>Subscribe j-th column</td>
</tr>
<tr>
<td><code>\$</code></td>
<td>Operator to add filter to a matrix</td>
</tr>
</tbody>
</table>
Function declaration & Scoping

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

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

```c
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.

```c
func rot90(double x, double y)
{
    return -y, x;
}
double x; double y;
x, y = rot90(x, y);
```
Return multiple values

```c
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
  \[ m_1 = m_2; \]  // their sizes do not have to agree.

- block assignment
  \[ m_1[x_{\text{low}}:x_{\text{high}}, y_{\text{low}}:y_{\text{high}}] = m_2; \]  // their size must agree.
  (so \[ m[:,:,:] \] is different from \[ m \] in assignment)
OpenCV related built-in function

- \[ m_r, m_g, m_b = \text{load(path)} \]
  - store image in RGB order
- \[ \text{save}(m_r, m_g, m_b, \text{path}) \]
  - save image defined by RGB matrices
- \[ m = \text{face(path)} \]
  - use OpenCV cascade classifier
  - \( m \) is a 4 by \( n \) matrix
  - \( n \): number 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)`
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