Language Proposal: “MMM”

Shenghao Jiang(sj2914)
Shikun Wang(sw3309)
Yixiong Ren (yr2344)

Role Responsibilities
Manager Timely (completion of deliverables) Shenghao Jiang
Language Guru (Language design) Yixiong Ren
System Architect (Compiler architecture, development environment) Shikun Wang
Tester (Test plan test suites) Shikun Wang

1.Description & Application of the Language

“MMM” is a Matlab style matrix manipulation language that can be used to calculate matrix operations efficiently. Users are able to do matrix operations through both in-built and self-defined functions. Same operations can be applied to user defined struct to make code more concise and more efficient. Due to our implementation and optimization of basic matrix operations in Ocaml, the basic matrix operations will be fast. The application of this programming language ranges from image cropping, rotating, denoising, enhancement, edge detection, and color filtering. Users are able to define the struct and functions to process image more efficiently.

2.Programs included

Within “MMM”, basic operators (such as +, -, *, /, …) need to be implemented to finish operations among integer, float and matrix. Besides, a built-in library should be established to calculate the properties of matrix such as determinant, eigenvalues, inverse, etc. For image processing additional functions need to be implemented to read images from files and save the post-processing images. A print() function is built as well to output results.

3. Parts

3.1 Primitive Types

int integer type
e.g.
```
int x = 1;
```
float float type
e.g.
```
float x = 9.012;
```
matrix matrix stores float type data
e.g.
```
matrix y = matrix(row,col);   // default to be all zero
matrix y = [2,2;2,3];
```

string quoted by “ ” and only used for image I/O & printing results

3.2 Lexical conventions

comments e.g.

```
// inline comments
/* multiple lines comments
*/
```

The language uses ; to separate each token of source code.
User must declare variable type before its name.
White space is required between the data type and the variable name, but it cannot appear in a variable identifiers or keywords.

3.3 Basic operators

+ addition
++ increase the integer by 1
- subtraction
-- decrease the integer by 1
* multiplication
/ division
== equivalence
!= non-equivalence
< less-than
> greater-than
<= less-than-or-equal-to
>= greater-than-or-equal-to
= assignment

3.4 Matrix Operator

+ matrix addition
- matrix subtraction
* matrix multiplication
.* element-wise matrix multiplication
./ element-wise matrix division

height(matrix): number of rows in the matrix as an integer
width(matrix): number of columns in the matrix as an integer
sum(matrix): sum of all the elements in the matrix
mean(matrix): average of all the elements in the matrix
trans(matrix): transpose matrix
eig(matrix): eigenvalues of matrix
inv(matrix): inverse of matrix

det(matrix): determinant of matrix
conv() convolution
conv(kernel, matrix) // convolve matrix with kernel

Slicing matrix e.g.

```
matrix M = [2,3,4,5;1,2,3,4;6,7,8,9];
M[row][col]
M[2][3]  //9
M[2][:] //[6,7,8,9]
M[:][2] //[4;3;8]
M[1:2][2] //[3;8]
```

3.5 Standard functions
declaration:

```
func name(arg1, arg2, ... ) {
}
```

call:

```
name(arg1, arg2, ...);
```

3.6 Program structures
If Else Statements
While statement
For statement

3.7 Struct
declaration:

```
struct name {
     type element1;
     type element2;
     ...
}
```

initialization:

```
name object =  name(arg1, arg2, ...);
```

call:
object.element

3.8 Image I/O

To conduct image processing, a built-in struct called Image is defined as follow:

```
struct Image{
    matrix R; // red channel
    matrix G; // green channel
    matrix B; // blue channel
    matrix BW; // white and black channel
}
```

Image I/O functions include:

```
Image img = imread("filepath"); // load image from file as a Image object
save(img, "filepath"); //save a Image object as a .jpg file
```

4. Sample Program

The sample image is a black and white image, and the purpose of this program is to denoise the image by assigning the kernel to calculate the mean of the neighbors of every pixel.

```
/*Image is a declared struct */
func noiseDeduction(Image sampleImage, int numOfIterations){
    matrix copyImage = sampleImage.BW;
    height = height(copyImage);
    width = width(copyImage);
    matrix kernel = [1.0, 1.0, 1.0; 1.0, 1.0, 1.0; 1.0, 1.0, 1.0];
    for(int i=0;i<numOfIterations;i++){
        for (int r=1;r<height-1;r++){
            for (int c=1;c<width-1;c++){
                matrix neighbor = copyImage[r-1:r+1][c-1:c+1];
                neighbor = neighbor .* kernel;
                float averageNeighbor =  mean(neighbor);
                copyImage[r][c] = averageNeighbor;
            }
        }
        sampleImage.BW = copyImage;
    }
    return sampleImage;
}
```