COMS 4115: NumNum Language Proposal

Programming Languages and Translators

COMS 4115 W Section 1

Prof. Edwards

September 26, 2017

Sharon Chen	syc2138	Tester
Kaustubh Gopal Chiplunkar	kc3148	Language Guru
Paul Czopowik	pc2550	Manager
David Tofu	dat2149	System Architect
Art Zuks	az2487	Tester

Abstract

We propose a matrix-manipulation language that allows basic arithmetic operations on arrays containing numbers. The main feature of the language is a Matrix object that contains built in features such as an index, bound checks, iteration, equality comparison and copy operation. The Matrix object would also include specific Matrix operations. More advanced algorithms such as sorting or searching are intended to be provided via libraries designed in our language.

The language also contains general features to allow building of such libraries and contains. These include primitive numerical data types such as integers and floats, flow control using functions, loops and decision if/then/else statements. Special keywords like _el(returning the current element), _next, _prev are introduced for quick access of special elements in arrays.

Language Purpose

The purpose of our language is to provide a native way to manipulate matrices and arrays. To make matrix manipulation easy, the language would feature simple syntax to allow basic matrix arithmetic, feature built in safety such as bound checking to prevent common errors and the ability to iterate through the data in the matrix.

An example of our language implementation could be to create programs that manipulate images. For example a program could be written to blur images or remove or adjust color information. Images are made of numbers arranged in matrices, which are multi-dimensional arrays of numbers. Because our language offers a native matrix interface it simplifies implementing libraries that would allow for image manipulation.

Basic Syntax

Core

```
# initialization
int a = [];
int b = [1];
int c = [2, 3];
int d = [[1, 2], [3, 4]];
int e = [c, d]; # e = [[2, 3], [[1, 2], [3, 4]]]

double g = [3.14];
bool f = [true];
```

```
func int foo() {
    return 3;
}
```

Arithmetic

```
# element-wise arithmetic
int a = [1, 3];
int b = [2, 4];
int c = a + b; # c = [3, 7]
c = a - b; # c = [-1, -1]
c = a * b # c = [2, 12]
double d = a / b; # d = [.5, .75]
```

```
# batch arithmetic
int a = [1, 3];
int b = [2];
int c = a + b; # c = [3, 5]
c = a - b; # c = [-1, 1]
c = a * b # c = [2, 6]
double d = a / b; # d = [.5, 1.5]
```

```
# matrix arithmetic
int a = [1, 3];
int b = [2];
int c = [a] + [b]; # c = [1, 3, 2]
c = [a] * [b] # c = [1, 3, 1, 3]
```

```
bool a = [true];
bool b = [false];
int c = [1];
int d = [2];

# boolean arithmetic
bool e = a == b; # e = [false]
e = a && b; # e = [false]
e = a || b; # e = [true]

# logical operators
```

```
e = c > d; # e = [false]
e = c < d; # e = [true]
e = c >= d; # e = [false]
e = c <= d; # e = [true]
e = c == d; # e = [false]
e = c != d; # e = [true]
```

```
# traversal
int c = [4,6,3,2,5];
double avg = [0];

#running average
iterate c {
   avg = (_el + avg * _pos-1)/_pos;
}
print avg; # 4
```

A Sample Case Study

Emily is working on an animation team at Pixar, and she intends to create a black-and-white scene that happens behind a window. She is given a sequence of color images that make up the scene. She has two goals: to turn each image in the sequence into black-and-white, and to blur each image in the sequence. As a result, she codes up the following programs, using our language because of its ease of use in matrix and image manipulation.

images.ppm (image manipulation library)

```
# traversal
int c = [4,6,3,2,5];
double avg = [0];
func int read_ppm(str fileName) {
     ppm_lines = readFile(fileName).lines()

     # Read dimensions, .split(" ") splits a string by spaces
     dim_x = ppm_lines[0].split(" ")[0]
     dim_y = ppm_lines[0].split(" ")[1]

     int ppm_mtx[dim_x * dim_y * 3] =iterate ppm_lines[2:] with row {
          int int_row = iterate row.split(" ") with word {
               return int(word) # Convert to int format
          }
          return int_row
     }

     # Reshape matrix, each pixel takes up three ints for RGB
     ppm_mtx.reshape(dim_x, dim_y, 3)

     return ppm_mtx
}
```

edit_movie.mm

```
# takes in a list of matrix representation of images

double movie = readFile "./coloredMovie.txt";
iterate movie {
   double image = _el;
   int idx = _i;
   writeFile "./coloredImage" + idx + ".txt";

   # do something with image

   image.print; # see resulting image
}
```

black_white.mm

```
# takes in an matrix representation of an image
# rounds down things greater than .5

double image = readFile "./coloredImage1.txt";
iterate image {
   if (_el <= .5) {
       _el = 0;
   } else {
       _el = 1;
   }
}

image.print;
```

blur.mm

```
# the keyword _surrounding when iterating an array
# return the elements adjacent to the current element
# while filtering out elements out of bounds
# _surrounding = [_top,_bottom,_left,_right]

double image = readFile "./matrixImage1.txt";
double blur = readFile “./blurImage.txt”;
iterate image blur {
     _el2 =  _el1.surrounding.avg();
}

image.print;
```

Bubblesort.mm

```
#Bubble sort to sort an array

int to_sort [20,15,6,4];

iterate in range 1:to_sort.length(){
     iterate to_sort {
          if (_el>_next){
                    temp=_el;
                    _el=_next;
                    _next=temp;

                    }
          }
}

to_sort.print();
```