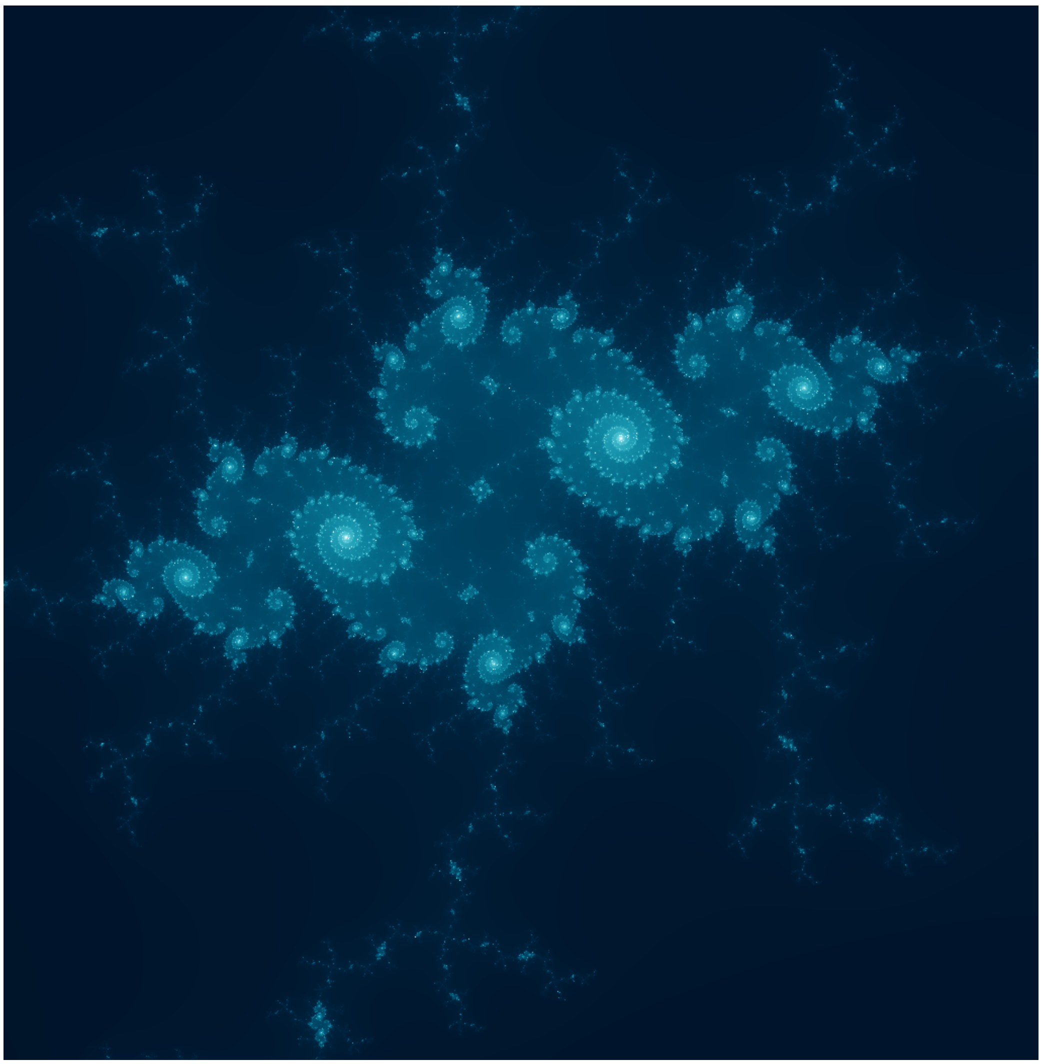
easel



Manager | Danielle Crosswell | dac2182
Language Guru | Tyrus Cukavac | thc2125
System Architect | Yuan-Chao Chou | yc3211
Tester | Xiaofei Chen | xc2364

INTRODUCTION & MOTIVATION

To most people mathematics is that pesky subject which you'll never quite understand; however, there's another side to math. Mathematics can be seen as an art form, like drawing or painting. Using easel, we'd like to do just that. Our ultimate goal of this language is to create art using math.

Being able to visualize mathematical formulas has multiple uses. The first is simply to create beautiful images with just a few lines of code. This language also provides an opportunity to view formulas in forms other than just the normal bar or line graph. easel will allow its users to visualize sets of data in ways that they haven't been viewed before. The user will be able to view their results in a more subjective manner, bringing creative possibilities to mathematics. easel also will enable the user to take functions and map them into an output that can be printed in 2D or 3D. Additionally, easel can be used to provide visualizations of projects such as mapping the internet, which involves a lot of data so it requires a way of mapping all of the data in an aesthetically pleasing manner. Finally, easel can be used to create fractals, which can only be drawn by computers. By combining the ability to draw interesting graphics and having the computational power of a computer, easel can effectively depict fractals.

LANGUAGE DESCRIPTION

The primary purpose of our language is to provide programmers a means of expressing data and functions as visual representations. As a result, the language is capable of performing both basic and advanced mathematical expressions as simple operators, without the use of additional libraries (for example, trigonometric and logarithmic functions). Primitive data types consist of some syntactic sugar to make the creation of images a central feature of the program. "canvas" and "pix" are data types representing the drawing canvas and pixel qualities, respectively. Vectors are a unique data type that can be used to assist in plotting points of data (similar to a graph). Additionally, because of the nature and role of functions in the creation of mathematically based images, functions are considered first-class objects. They can be passed as parameters, returned as values, and declared anonymously in pursuit of those objectives. In order to allow for robust execution of languages, basic timer functionality has also been baked into the language, allowing a programmer to define when given functions should execute.

BASIC DATA TYPES

TYPE	CONTAINS	SYNTAX	NOTES
bool	boolean value	bool *VAR* = *VALUE*;	value stored as 0 or 1 and can be represented by "true" or "false" keywords
int	numeric value	int *VAR* = *VALUE*;	can be input using either hex or decimal notation
float	floating point numeric value	float *VAR* = *VALUE*;	can be input using decimal notation
string	string value (array of characters)	string *VAR* = *STRING- VALUE*;	generally only used to specify filepaths
pix	numeric value	pix *VAR* =*24-BIT-VALUE*; pix *VAR* [red/green/blue]=*8-BIT-VALUE*;	can be input using hex, decimal, or specialized array notation

PIX DATA TYPE

A pix essentially stores a color value from 0-16777216 (i.e. 6 hex values) in order to define colors for a specific pixel. pix can use either standard integer/hex notation, or can be accessed similarly to a map (red being a designated keyword that extrapolates the value of red from the pixel's overall numerical value). For example:

    pix myPix[red]=#ff

Pixels can be thought of as a combination of char and string types in other languages: char because each pixel is a single numerical value, string because syntactically the pixel can be altered one color at a time.

A pixel can be defined by passing it a list of the form {#red, #green, #blue}

Example:

```
pix redPix = #ff0000;
pix redPix2 = 16711680;
pix redPix3 = {255, 0, 0};
pix redPix4[red] = 255; /* This would set the pixel to have a maximum red value.
                           Green and blue would automatically be set to 0 and the
                           pixel integer value would equal #ff0000 */
```

COMPOUND DATA TYPES

TYPE	CONTAINS	SYNTAX	NOTES
array	a collection of primitive elements	*PRIMITIVE_TYPE VAR*[*ITEMS*];	each element within the array must be defined separately
matrix	a collection of arrays	*PRIMITIVE_TYPE VAR* [*ITEMS*][*ITEMS*]	
tuple	a collection of elements	*PRIMITIVE TYPE VAR* = {*ITEM_1*,*ITEM_2*, .. .,*ITEM_N*};	items can be added or removed from tuples using special function
canvas	syntactic sugar; essentially a matrix of pixels.	canvas *VAR* [*SIZE_X*][*SIZE_Y*];	both rows and columns of a canvas element are 2x+1 of the user provided size and begin from -*SIZE* to +*SIZE* (to approximate a cartesian graph with a center index of (0,0))
vector	syntactic sugar; a 2-tuple with ordered pairs of coordinates	vec *VAR* = {{0,1},{1,2}};	two-tuples can be passed as values as well as constants.

Negative indexing is allowed to access array and matrix elements, by counting from the tail. For example, given an array a of size 10, a[-8] is equivalent to a[10 - 8] or a[2].

Lists, arrays, matrices, and canvases have the property "size" which indicates the length of the list/sublist/array/matrix.
Example: canvas c[10][20]; canvas[2].size = 20

MATHEMATICAL OPERATORS

OPERATOR	MEANING	UNARY	SYNTAX
+	Addition	++	int a = 5+5;
-	Subtraction	--	int a = 5-5;
/	Division	//	int a = 5/5;
*	Multiplication	**	int a = 5*5;
%	Modulus	%%	int a=5%5;
^	Exponentiation	^^	int a=5^5;
~	Sine		float a=~5;
~~	Cosine		float a=~~5;
@	Tangent		float a=@5;
|	Logarithm		float a=5|5; /* base 5, log 5 */ float b=|5; /* defaults to base 10 */
$	Root		float a=3$64; /* cube root of 64 */ float b=$16; /*defaults to square root */

OTHER OPERATORS

OPERATOR	MEANING
=	assigns a value to a variable
<	less than
>	greater than
==	equivalent
<=	less than or equal to
>=	greater than or equal to
!=	not equivalent to
&&	logical AND
||	logical OR
!	NOT
.	access operator (accesses value of *KEY* in *OBJECT.KEY*)
/* **...** */	commenting

CONTROL FLOW

SYNTAX	MEANING
if (*EXPR*) {*STATEMENT*} **else {*STATEMENT2*}**	if *EXPR* evaluates to true execute *STATEMENT* otherwise evaluate *STATEMENT2*
for (*STATEMENT1; EXPRESSION; STATEMENT2*) { *STATEMENT3;}*	repeat *STATEMENT3* until *EXPRESSION* evaluates to false
do { *STATEMENT; } while (EXPRESSION*);	repeat *STATEMENT* until *EXPRESSION* evaluates to false
while (EXPRESSION*) { *STATEMENT; }	repeat *STATEMENT* until *EXPRESSION* evaluates to false

FUNCTIONS

The basic grouping of statements in easel - every body of code is considered a function. Functions can be passed as arguments to other functions and can be called recursively.

function *RETURN_TYPE NAME*(*PARAM1*, *PARAM2*) {
    STATEMENTS
}

Functions as a parameter:

call(*PARAM1*, function *RETURN_TYPE*(*PARAM1*, ... , *PARAMN*) {*STATEMENTS*})

Functions as a return value:

function *RETURN_TYPE FUNCTION1* (*VAR*) {
    return function *RETURN_TYPE* (*PARAMS*){ *STATEMENT WITH VAR;* }
}

SPECIAL FUNCTIONS

Function Call	Description	Notes
void draw(int *x*, int *y*)	draws a given canvas to the screen	can only be called by canvas elements; integers represent the top corner of the image; non-blocking function - the canvas will remain of the screen as following statements execute
void drawout (int *overwrite_time*, string *file_path*)	saves file to given filepath	can only be called by canvas elements; if the gif file already exists and integer is 0 or higher, the function will draw the canvas as an additional frame into the gif, placed at the time period of the given int
canvas view (string *file_path*)	scans in a given gif file and returns a canvas element	
void graph (function *black*) **void graph (function *shape*, function *color*)** **void graph (function *red*, function *green*, function *blue*)**	overloaded function graphs a given function onto a canvas	can be called by canvas elements; accepts up to 3 functions as arguments as well as an int value representing the axis to map the function to; default color to map to is black, can specify with other functions the colors of each pixel
void execInterval (int *t*, function *f*)	function that executes a given function over an interval t	integer value represents an integer in milliseconds; allows for basic animation
void execAfter (int *t*, function *f*)	function that executes a given function after a period of time t	integer value represents a time period in milliseconds

SAMPLE CODE

Draws the Mandelbrot set displayed on the cover page:

```
/* mandelbrot.es */
/* mapping pixel position to intensity of red */
function int mandelbrot_red(int col, int row) {
    float a = 0, b = 0, c, d, n = 0;
    while ((c = a * a) + (d = b * b) < 4 && n++ < 880) {
        b = 2 * a * b + row * 8e-9 - .645411;
        a = c - d + col * 8e-9 + .356888;
    }
    return (int) (255 * ((n - 80) / 800) ^ 3);
}

/* mapping pixel position to intensity of green */
function int mandelbrot_green(int col, int row) {
    double a = 0, b = 0, c, d, n = 0;
    while ((c = a * a) + (d = b * b) < 4 && n++ < 880) {
        b = 2 * a * b + row * 8e-9 - .645411;
        a = c - d + col * 8e-9 + .356888;
    }
    return (int) (255 * ((n - 80) / 800) ^ .7);
}

/* mapping pixel position to intensity of blue */
function int mandelbrot_blue(int col, int row) {
    double a = 0, b = 0, c, d, n = 0;
    while ((c = a * a) + (d = b * b) < 4 && n++ < 880) {
        b = 2 * a * b + row * 8e-9 - .645411;
        a = c - d + col * 8e-9 + .356888;
    }
    return (int) (255 * ((n - 80) / 800) ^ .5);
}

function int main() {
    canvas mycnvs[1024][1024];

    /* provides functions as arguments for graphing,
    then draws picture on center of screen,
    finally outputs picture to gif file named mandelbrot.gif */
    mycnvs.graph(mandelbrot_red, mandelbrot_green, mandelbrot_blue);
    mycnvs.draw((1920 - 1024) / 2, (1080 - 1024) / 2);
    mycnvs.drawout(0, “mandelbrot.gif”);
}

/* call the entry function and return its result */
return main();
```