pixelman

Pixel Manipulation Language

Anthony Chan tc2665
Teresa Choe tc2716
Gabriel Lucas Kramer-Garcia glk2110
Brian Tsau bt2420

1. Introduction/Rationale

pixelman is a programming language that can be used to manipulate pixels to edit image files. When brainstorming for our programming language, we thought about specific problems our language could be used to solve. In the case of image processing, we thought about how to manipulate pixels through functions; for example, by applying a convolution matrix to pixels, we could sharpen or blur an image. By allowing users to include their own tools to perform these operations, we allow them to create powerful image processing programs.

2. Project Description

pixelman will provide tools to load/write images to/from a filesystem, I/O operations with stdin and stdout, and image manipulation functions. Compiled code will be translated into LLVM. Our most powerful feature will be the pixel transformation tool, which will take in a function that takes in a pixel and outputs a pixel, and can be used to perform a variety of image manipulations.

3. Language Features

A. Data Types

Primitive data type	Description
int	32-bit signed integer
float	32-bit floating point
string	List of characters enclosed in double quotes

boolean	true/false
null	null

B. Conditionals

Conditional	**Syntax**	**Description**
if, else, else if	if(expr) {} else if {} else {}	Executes block of code if condition is met.
for	for(int i = 0;i < 8;i=i+1) {}	Loops until condition (i < 8) is met.
while	while(expr) {}	Loops while expression evaluates to true.

C. Operations

Operator	**Syntax**	**Description**	**Return Type**
=	a = b	Assigns variable to expression of same type (or ints to floats)	Type of a
+	a + b	Addition, if an argument is a float will return float	int, float
-	a - b	Subtraction, if an argument is a float will return float	int, float
*	a * b	Multiplication, if an argument is a float will return float	int, float
/	a / b	If both arguments are	int, float

		integers, will perform integer division; else floating point division	
%	a % b	Modulus Operator. Returns remainder of a/b. a and b must both be ints.	int
==	a == b	Checks to see if variables are equal to each other	boolean
!=	a != b	Checks to see if variables are not equal to each other	boolean
>=	a >= b	Checks to see if argument on the left side is greater than or equal to argument on right	boolean
<=	a <= b	Checks to see if argument on the left side is greater than or equal to argument on right	boolean
!	!a	Boolean NOT	boolean
&&	a && b	Boolean AND	boolean
||	a || b	Boolean OR	boolean

<<	a << b	Left bit shift, b must be an integer	type of a
>>	a >> b	Right bit shift, b must be an integer	type of a

D. Syntax

pixelman will have a similar syntax and aesthetic to Java, and functions will be treated the same way as variables (as in Python). pixelman is strongly typed, and in function declarations, argument and return types must match up at compilation time.

1. **Complete statements**
   Semicolons will be used to declare the end of a statement
2. **Comments**
   There will only be single line comments, "//"
3. **Blocks of code**
   Blocks of code will begin with a "{" and end with a "}"
4. **Function declaration**
   Functions will be declared "def type functionName(arguments) {code}". A main function is necessary for execution of code.
5. **Whitespace**
   All whitespace and newlines will be processed out and used to separate tokens
6. **Reserved Keywords**
   return, def, main, all data types
7. **Code Example**

```
def int greyscale(string file_name) {
      if(image im = load(filename) == null) {
            perror("original image did not load");
      }
      printf("original image:");
      display(im);
```

```
        for(int i = 0; i < size(im); i++) {
            for(int j = 0; j < size (im[i]); j++) {
                int[] rgb = im[i][j].RGB;

                // Calculates grayscale by calculating
                the average

                int grey = (rgb[0] + rgb[1] +
                rgb[2])/3;

                im[i][j].RGB = int[3]{grey, grey,
                grey};
            }
        }
        printf("grayscale image:");
        display(im);
    }

    def int main(args, argv) {
        string file_name = 'admin/image.jpg';
        greyscale(file_name);
    }
```

E. <u>Data Structures</u>

Data Structure	**Syntax**	**Description**
List	`int[3] arr = {1, 2, 3};` `arr[0] = 0;`	Mutable data structure holding multiple blocks of information of the same type
Pixel	`pixel pix = pixel(r, g, b)`	Data structure with list of three integer values (RGB) capped in the range of 0-255, and two integers (x, y) coordinates

Image	image im = load(file_path) im.	Contains a image height × image width list of pixels

F. Built in Library

Function	**Arguments**	**Description**	**Return Type**
printf	string format, ...	Prints to stdout, using a similar model to C	On success: # characters written On failure: -1
scanf	string buffer	Scan from stdin	On success: # characters scanned On failure: -1
size	list arr	Gets length of list	On success: int On failure: -1
load	string file_path	Loads image from file	On success: image On failure: null
write	string file_path, image image	Will write image to file	On success: 0 On failure: -1
display	image im	Shows image without saving	On success: 0 On failure: -1
resize	image im, int w, int h	Resize image to new size w × h	On success: image On failure: null
transform	image im, function process	Performs an operation defined by the user in function on each pixel in image, and creates a new image	On success: image On failure: null