WARHOL

Martina Atabong
Manager
maa2247

Charvinia Neblett
Language Guru
cdn2118

Sarina Xie
Compiler Architect
sx2166

Samuel Nnodim
Develop Environment Architect
son2105

Catherine Wes
Tester
ciw2109

1 Introduction

Warhol is a functional programming language that allows the user to easily manipulate images. Images are uploaded from files and parsed into a matrix capable of holding the original pixel information. The goal of our language is to make it very easy for users to manipulate images, inspired by many of the functionalities of Photoshop. The user will upload an image locally and is able to change the image by implementing a number of built-in functions or supported action. Key features include:

- Image Replication
- Image Color Filtering
- Image
- Sharpening and Blurring
- Size manipulation
- Isolated Color Filter

2 Language

The Warhol compiler will be written completely in OCaml, and compiled with LLVM. The structure of the language is function with an emphasis on our main primitive type – matrix. The dynamic use of a matrix, designed to store integer tuples, allows Warhol to do frequent writes, reads, and pixel manipulation.
The language is syntactically similar to Java with inspiration for matrix manipulation from MatLab, and libraries similar to C. Built-in functions are provided to compute commonly used image algorithms while also giving users freedom to implement functions.

3 Syntax

3.1 Comments

Comments using open and closed #'s will denote a section commented out.
Ex: #open comment ……. close comment#

3.2 Data Types

a. Pixel
   Declaration: pix test(R, G, B)
   Access: test(index); #index is 0 for Red, 1 for Green, and 2 for Blue#
   Note: Default value is 0 if not a complete tuple
b. Matrix
   Declaration: mat test = [12 62 93 -8 22; 16 2 87 43 91; -4 17 -72 95 6];

```
test =
   12   62   93   -8   22
   16    2   87   43   91
   -4   17  -72   95    6
```

   Access: mat newTest = test(0:1;2:4);

```
newTest =
   93   -8   22
   87   43   91
```

   Note: All matrices must be rectangular
c. Integer
   Declaration: int test = 4;
d. Booleans
   Declaration: bool test = true;

3.3 Function Declaration

Declaration: fun test (a, b, c) = {code inside the function};

3.4 Relational Operators

a. is-equal-to:
   expression_a == expression_b
b. is-less-than:
   expression_a < expression_b
c. is-greater-than:
   expression_a > expression_b
d. is-not-equal-to:
   expression_a != expression_b
e. is-less-than-or-equal-to:
   expression_a <= expression_b
f. is-greater-than-or-equal-to:
   expression_a >= expression_b

3.5 Boolean Operators

a. OR: ||
b. AND: &&
c. NOT: !

3.6 Mathematical Operators

a. Add : expression _a + expression _b
b. Subtract: int_a - expression_b
c. Divide: expression_a / expression_b
d. Multiply: expression_a * expression_b
e. Note: all integer operations, matrix follows matrix multiplication, pixel is only valid for adding and subtracting RGB respectively

3.7 Built-in functions

a. read() – Read/upload an image
b. save() – Save an image
c. filter() – Filter image
d. reverse() – Reverse matrix (for flip)
e. replicate() – Replicate image function
f. display() – Submit input function for display

3.8 Reserved Words

a. Boolean literals: true false
b. types: pix, mat, int, save, read

3.9 Control Flow

a. For and while loops for iterating
b. If, else if, and else statements

3.10 Miscellaneous

a. Whitespace is ignored
b. Statements are concluded by semi-colons

4 Example Code

4.1 Read, Replicate, and Display Image

```
mat image = read('picture.ppm'); #read in an image into a matrix#
mat image2 = replicate(image,[3;5]); #replicate the image 15 times in a 3x5 matrix#
save(image2); #displays the replicated image on a new page#
```

4.2 Filter

```
mat image = read('picture.ppm'); #read in an image into a matrix#
filter(image, (R,G,B), opacity); #built in filter image function#
```

4.3 Horizontal Flip

```
mat image = read('picture.ppm');
mat flippedImg;
for (i = 0; i<numrow(image)-1; i++) {
        row = image(i:i+1);
        newRow = reverse(row);
        flippedImg = flippedImg + newRow;
}
```

4.4 Vertical Flip

```
mat image = read('picture.ppm');
mat flippedImg;
for (i = 0; i<numcol(image)-1; i++) {
        col = image(i:i+1);
        newCow = reverse(col);
        flippedImg = flippedImg + newRow;
}
```