Colode

“Bringing color to code”

Steven Bonilla (sb3914@columbia.edu) - Manager
Tahmid Munat (tfm2109@columbia.edu) - Language Guru
Takuma Yasuda (ty2353@columbia.edu) - Language Guru
Willie Koomson (wvk2101@columbia.edu) - System Architect
Dimitri Borgers (djb2195@columbia.edu) - Tester

1. Introduction

Colode is an efficient programming language used for manipulating images. It includes built-in functions and types to crop, rotate, scale, and manipulate individual or groups of pixel values. With these tools, in addition to a syntax similar to Python, this language provides an extremely easy-to-learn and advanced way to modify image files for any use.

1.1 Implementation

Colode will be designed to access image files available in the directory (with the use of coload, and coclone functions), modify these files (using cozoom, among many other functions) and then compiling the eventual changes with the LLVM. With a strong emphasis on readability like the Python syntax, minus a few features considered, such an indentation, and libraries similar to C, this entire process will be seamless to implement. The built-in functions available can be used for the most common editing features available on other expert platforms.

1.2 Use Cases

The number of available softwares to edit pictures is now in the many hundreds; each with their own learning curve and correlated ability to revise an illustration. None provide an accessible interface and high-level features. Colode eliminates this inconsistency. With the ability to control individual pixels on a loaded image file, a user will be able to alter anything at the most detailed level. In addition, by grouping pixels together, the process of editing will also become much more convenient and less time consuming. Photoshop softwares currently available don’t allow for both precise and broad editing.

2. Language Overview

2.1 Operators

Type	**Description**
Arithmetic	+, -, *, /, ^,%
Logical	and, or, not
Comparison	<, >, <=, >=, ==, !=
Assignment	=, +=, -=, *=, /=
Comments*	//

* **Comments** are only for one line. Finish at each endline.

2.2 Data Types

Primitive	**Description**
int	Basic integer type, 4 byte in size
float	Floating point numbers, 4 bytes in size
bool	8-bit boolean variable. True or False.
char	ASCII characters, 1 byte in size

Compound	**Description**	**Syntax**
list	Unordered, iterable collection of single-type data. Indexed using bracket notation.	[‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
string	Immutable null-terminated list of chars.	Arg_name = “hello”

2.3 Built-in Types

Type	**Description**
Image	Holds pixel values in 2D array. Can be accessed used image[width][height]). **Properties:**

	- *int* width - *int* height
Pixel	RGB, 3-vector of floats **Properties:** - *float* r - *float* g - *float* b
Matrix	Data structure storing bool/ints/doubles of arbitrary size

2.4 Control Flow

Conditionals	**Description**	**Syntax**
If, elif, else	Zero or more “else if”statements allowed. “Else” addition is optional.	if condition: (statements) else if condition: (statements) else: (statements)
Return*	Terminates the execution of a function and returns control to the calling function.	return

***Return** function gives None value if function has no return

Loops	**Description**	**Syntax**
For loops	Iterates through statements after colon and between parentheses until told to exit based on a condition within the loop	for var in condition: (statements) for var in range(int, int): (statements)
While loops	Repeatedly executes a target statement as long as a given condition is true.	While condition: (statements)
Break	Breaks the control flow out of loop	break
Continue	Returns the control to the beginning of the while loop	continue

2.5 Keywords

Keyword	**Description**	**Syntax**
def	Function declaration. Useful for user-defined functions.	def add(x, y): sum = x + y return sum
range	Provide range between two integer values	range(int, int)

2.6 Built-in functions

Function	**Description**
print()	print function
coload(filename)	load an image; returns a file descriptor (int)
coclone(int)	clone and identical copy of the source image
cocrop(int, int, int)	crop an image; arguments take a percent value for width and height
cozoom(int, int)	zoom an image; argument takes a percent value to zoom in/out
corotate(int, char)	rotate an image; argument only takes c (clockwise) or u (counter-clockwise)
coscale(int, int, int)	scale an image; arguments take the image, width, and height
coclose(int)	close the file

2.7 Indentation and Whitespace

Indentation does not affect syntax.
Whitespace is ignored when compiling

3. Sample Program

3.1 Add a desired signature to a picture

```
signImage()
def signImage():
      // Load an image and create copies for different operations
      img = coload(“image.png”)
      imgCopy = coclone(img)
      rotateCopy = coclone(img)
      zoomCopy = coclone(img)
      cropCopy = coclone(img)

      // Load a signature and create a copy
      sign = coload(“signature.png”)
      signCopy = coclone(sign)

      // Close source image files
      coclose(img)
      coclose(sign)

      // zoom the image by 50%
      cozoom(rotateCopy, 50)

      // crop the image (10% width, 20% height)
      cocrop(cropCopy, 10, 20)

      // rotate the image clockwise and counter-clockwise
      corotate(rotateCopy, c)
      corotate(rotateCopy, u)

      // calculate the width and height for the signature
      s_width = imgCopy.width * 0.05
      s_height = imgCopy.height * 0.05

      // scale the signature image if necessary
      if signCopy.width > s_width or signCopy.height > s_height:
            coscale(signCopy, s_width, s_height) // built-in

      // copy over the sign
      for i in range(0, s_width):
```

```
      for j in range(0, s_height):
            imgCopy[imgCopy.width - s_width + i][imgCopy.height -
            s_height + j] = signCopy[i][j]

// close copied image files
coclose(imgCopy)
coclose(signCopy)
coclose(rotateCopy)
coclose(cropCopy)
coclose(zoomCopy)
```