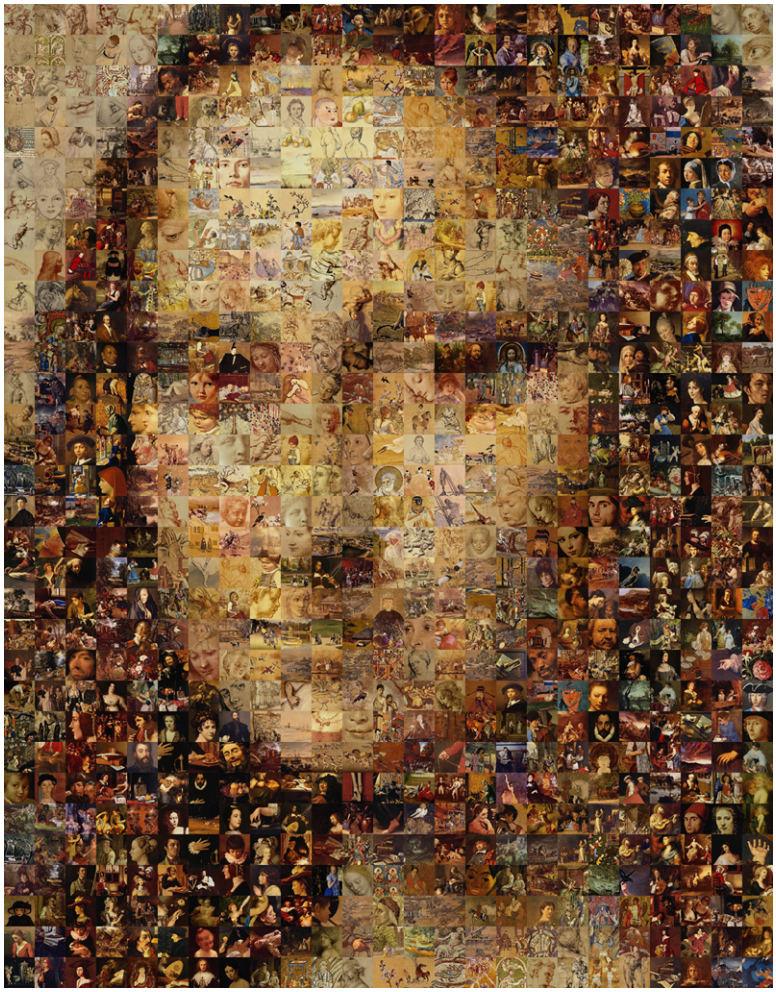


Photogram

Programmable Photoshop® Language

Language Reference Manual

Ohan Oda **(oo2116@columbia.edu), Group Leader**
Neesha Subramaniam **(ns2295@columbia.edu)**
Richard Ng **(rjn2003@columbia.edu)**
Seikwon Kim **(sk2617@columbia.edu)**

Table of Contents

1. Introduction

Adobe® Photoshop®, the professional image-editing standard and leader of the Photoshop digital imaging line, has been the most prominent and powerful image-editing tool over the years. Although the software is very powerful, due to its complexity and overwhelming graphic user interface, it is difficult for the users to achieve the results they want without being very experienced. Additionally, even though Photoshop® provides thousands of image-editing functions such as resizing and filtering, the capabilities of image manipulation are limited to what the software provides (e.g. the software provides Gaussian blur, motion blur, radial blur, and smart blur for blurring operations, but the user cannot use their own specified blurring filter). Further more, undoing or fixing previous undesired operations is a tedious task using Photoshop®. One common occurrence is that a user performs several different operations on an image and finds out later that the operation step 3 is undesirable after completing operation step 10. Now he/she is forced to cancel/undo all of operations down to step 3, and then redoing all of the steps from 4 to 10 after either fixing or eliminating step 3. Another example of a tedious task in Photoshop® is making multiple collages consisting of multiple images because applying the same operations on multiple images for multiple collages is extremely repetitive. For a collage, the user is required to cut and paste from many windows that contain images and it can easily reach the point where finding the desired image is a chore. A repetitive task such as applying the same blurring filter on an entire directory of images also requires the user to go through the same operation for each image which is an inefficient time sink.

Thus, it will be very beneficial if there is a language that can perform image-editing functionalities. The language will solve all of the existing issues on Photoshop addressed above. It will significantly save the time of people performing tedious image editions.

2. Lexical Conventions

2.1 Comments

Photogram creates comments in the same style as C++ or Java®.

2.1.1 Multiple Line Comment

To create a comment spanning multiple lines, use “/*” as the start of a multiple line comment and “*/” to terminate the comment.

2.1.2 Single Line Comment

For single line comments, use “//” to denote the beginning of the comment and the comment is the text following “//” until a new line character is reached.

```
Example
//This is a SINGLE LINE COMMENT
/* This is a
   MULTIPLE LINE COMMENT */
```

2.2 Identifiers

An identifier is a sequence of letters or digits and the first character of the identifier must be a letter from a to z or A to Z. The identifier cannot begin with a digit or any types of punctuation, and there is no length constraint on the identifier.

2.3 Keywords

The following list of words contains keywords, or reserved words of Photogram. These keywords cannot be used as ordinary identifiers. They must be spelled exactly as follows.

if	*false*
else	*continue*
while	*break*
switch	*void*
case	*return*
for	*null*
true	*new*

2.4 Constants

There are three kinds of constants as follows.

2.4.1 Integer Constant

An integer constant is simply a sequence of digits. There are no fractions or exponentials allowed.

2.4.2 Double Constant

A double constant consists of three sections, an integer part, a decimal point followed by the fraction, and an ‘e’ followed by an optionally signed integer exponent.

2.4.3 String Literal

A string literal is simply a sequence of characters, digits, or symbols. The beginning of the literal is denoted by the double quote, ' " ', and is also terminated with the same double quote.

```
Example.
Integer: 123
Double: 23.21e+21
String: "This is a string in \"Photogram\" for PLT class."
```

2.5 Other Tokens

Symbolic characters and sequences of symbolic characters are also used. They must be specified exactly as follows.

NAME	SYMBOL
Brace	{ }
Bracket	[ ]
Parentheses	( )
Multiply	*
Divide	/
Plus	+
Minus	-
Modulo	%
Assignment	=
Plus Equal	+=
Minus Equal	-=
Multiply Equal	*=
Divide Equal	/=
Modulo Equal	%=
Equal	==
Not Equal	!=
Greater than or Equal	>=
Less than or Equal	<=
Greater than	>
Less than	<
NOT	!
AND	&&
OR	||
Bitwise-operator OR	|
Bitwise-operator AND	&
Bitwise-operator Exclusive OR	^
OR Equal	|=
AND Equal	&=
Exclusive OR Equal	^=
Left Shift	<<
Right Shift	>>
Plus Plus	++
Minus Minus	--

Macro	#
Concatenation	##
Semicolon	;
Comma	,
Period	.

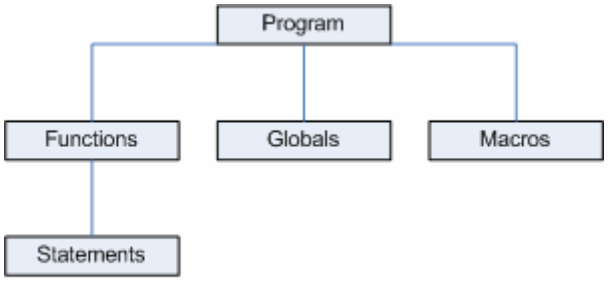
3. Input and Output of Photogram

Given a Photogram program, the input into the program are images which are specified as an argument before running the program and the output of the program will be also images that have been modified or created by the program as shown in the figure below.

Inputs and Outputs of a Program

4. Program Structure

A Photogram program consists of three components, functions, globals and macros as depicted in the figure below. Any one of these of these components may be present in the program or none at all. There is no limit to the number of any component present in a Photogram program.



Program Structure of Photogram

4.1 Functions

Functions are the fundamental building blocks of Photogram and are essentially blocks of statements that are executed in order. Functions take in a list of variables that are specified by data types and return a value of a specified data type. Functions must be specified in the following manner:

<data_type> <identifier> (<var_list>) { <statements> }

Elements enclosed within < and > should be replaced with the appropriate components.

There must be one function that must be in every program and that is the main function which is executed first whenever a program is run. This main function can return an integer or be void, but it must also contain certain parameters as shown below:

<int or void> main(String[] args)

4.1.1 Statements

A statement is a single instruction performed by Photogram which could include many types of statements varying from assignment to control flow of the program. They are usually executed in sequence unless a control flow statement is executed at which point the program could branch.

4.1.1.1 Selection Statement: if, else, switch, case

These statements select a flow of control for the program and the expression must evaluate to true or false. The selection statements must follow the following format:

if (expression) statement

if (expression) statement1 else statement2

```
switch (primitive)
{
case <primitive>: <statement> break;
...
default: <statement>;
}
```

4.1.1.2 Looping Statements: For, While

These statements are used for looping through a block of statements. The expressions used in while loop and the second condition of the for loop must evaluate to true or false and the expression in the third condition of the for loop must be an increment or decrement of some kind. The loops must follow the following format:

```
for (<assignment>; <expression>; <expression>)
        {
        <statements>
        }

while(<expression>)
        {
        <statement>
        }
```

4.1.1.3 Break statement

When this statement is executed, it immediately breaks out of the current loop that it is in and goes executes the next statement. The statement is simply used as:

break;

4.1.1.4 Continue Statement

When this statement is executed in a loop, it ignores the rest of the loop and simply begins the next iteration of the loop and the statement is used as:

continue;

4.1.1.5 Function Call Statement

This statement is used to call a function that has been defined elsewhere and pass into the function the required parameters. This is accomplished in the following manner:

<function identifier>(<expression list>);

4.1.1.6 Return Statement

This statement is used in a function in order to end the function and return an expression to the statement that called the function. The expression must evaluate to the same data type as the function. The return statement is used as so:

return <expression>;

4.1.1.7 Assignment Statement

This statement is used to assign a value to an identifier. The data type of the right side must be the same as the data type on the left side. The assignment statement is written as shown below:

<identifier or array> <assignment operator>
<expression>;

<identifier or array> = <function call>;

4.1.1.8 Primitive Declaration Statement

This statement shows how to declare a primitive:

<primitive> <identifier>;

4.1.1.9 Built-in Object Declaration Statement

This statement shows how to declare a built-in object:

<built-in object> <identifier> = new <built-inobject>(<expression list>);

4.1.1.10 Array Declaration Statement

<data type>[] = new <data type>[<number>];

4.2 Globals

Globals are used to declare global variables that can be accessed from any function. It follows the same format as the declaration or assignment statements in a function.

4.3 Macros

Macros are used as a method of preprocessing in including files with additional functions or defining constants.

4.3.1 Include Macros

#include <file name>

4.3.2 Define Macros

#define <identifier> <constant>

5. Scope

Photogram uses static scoping exactly the same as Java® or C++.

6. Namespace

6.1 Function Namespace

This namespace is reserved for the names of functions used in Photogram.

6.2 Variable Namespace

This namespace is reserved for the names of variables used within functions and the name of global variables.

7. Data Types

Photogram requires programmers to explicitly specify the data types for each variable they use. There are two different data types, primitives and built-in objects. Photogram supports a number of primitives just like other programming

languages, but only primitives useful for image manipulation were selected. There are also a number of built-in objects that are necessary for image manipulation however the programmer is not given the option of creating additional objects.

Photogram does not give the option for programmers to explicitly cast any data types. Photogram will automatically cast (implicit casting) a data type depending on what is the expected data type. (e.g. for passing parameters, if the parameter's type is integer and the user passes a double integer, Photogram will automatically converts it to an integer; for assignment statement, if the left value's data type is double integer and the right value's result is an integer type, then Photogram will automatically converts it to a double integer). This limitation is to avoid crashes caused by incorrect casting (e.g. casting integer type to an Image object,
Image i = (int)a).

7.1 Primitives

Photogram support four types of primitives: *int*, *double*, *boolean*, and *void*.

7.1.1 int

32-bit integer

7.1.2 double

Double-precision (64-bit) IEEE floating point number

7.1.3 boolean

Boolean value: either ***true*** or ***false***

7.1.4 void

This is only used as a return type for functions that do not return a type

7.2 Built-in Objects

Photogram supports eight types of built-in objects: Line, Color, Pixel, Rect, Oval, String, Image, and Font. (Note: All of the example function calls mentioned below will be implemented using Java as our built-in library functions)

7.2.1 Line

A line with a start point and an end point

```
Line line = new Line(int sx, int sy, int ex, int ey);
drawShape(line, color); // draw a line with a color
```

7.2.2 Color

A color with RGBA value; each Red, Green, Blue, and Alpha value is in the range between 0 and 255

Color color = new Color(int r, int g, int b, int a);

7.2.3 Pixel

A pixel value with x-y position and a color

Pixel pixel = new Pixel(Color color, int x, int y);
image.setPixel(pixel); // replaces the pixel color at (x, y) in image

7.2.4 Rect

A rectangle with upper left corner point, width, and height

Rect rect = new Rect(int x, int y, int width, int height);
drawShape(rect, color); // draw a rectangle with a color

7.2.5 Oval

An oval with center point and horizontal and vertical length

Oval oval = new Oval(int x, int y, int a, int b);
drawShape(oval, color); // draw an oval with a color

7.2.6 String

A sequence of characters specified inside of two double quotation marks "…".

String string = { new String("hello") or just "hello"}
drawText(string, x, y, font, color); // draw a text at starting location (x, y) with a specific font and color

7.2.7 Image

An image with an array of 32-bit integers

Image image = new Image{(int width, int height) or (int [][] pixels)}
image.saveAs("new_image", image.JPEG_FORMAT);

7.2.8 Font

A font with font family constant, font size, and special effect constant (BOLD, ITALIC, UNDERLINE, EMBOSS, etc)

Font font = new Font(Font.ARIAL, 12, Font.PLAIN);

8. Expressions: Operators and Precedence

The precedence of operators in the order of decreasing precedence is described as follows. We support comma separated expressions in Photogram.

8.1 Parentheses

The user can override the precedence rules of Photogram by using parentheses around the expression.

8.2 Not, Increment, and Decrement Operators: !, ++, --

8.2.1 Not Operator

The logical not operator inverts the value of the expression, zero being inverted to one and vice-versa.

8.2.2 Increment Operator

The ++ operator increments the left-value by 1.

8.2.3 Decrement Operator

The -- operator decrements the left-value by 1.

8.3 Multiplication, Division, Modulo Operators: *, /, %

8.3.1 Multiplication Operator (*)

The operator '*' multiplies the two numeric tokens.

8.3.2 Division Operator (/)

The operator '/' divides the first operand with the second operand.

8.3.3 Modulo Operator (%)

The operator '%' provides the remainder when the first operand is divided by the second operand.

8.4 Addition and Subtraction Operators: +, -

8.4.1 Addition Operator

The '+' operator computes the sum of the first and second operand.

8.4.2 Subtraction Operator

The '-' operator subtracts the value of the second operand from the first operand.

8.5 Shift Operators: >>, <<

8.5.1 Left Shift Operator
The "<<" operator shifts the bits of the first operand left by the number of bits specified by the second operand.

8.5.2 Right Shift Operator
The ">>" operator shifts the bits of the first operand right by the number of bits specified by the second operand.

8.6 Relation Operators: <, <=, >=, >

8.6.1 Less than Operator
The '<' operator evaluates whether the first operand is less than the second operand and returns a Boolean value 1 or 0 as the output.

8.6.2 Less than or Equal to Operator
The "<=" operator evaluates whether the first operand is less than or equal to the second operand and returns a Boolean value 1 or 0 as the output.

8.6.3 Greater than Operator
The ">" operator evaluates whether the first operand is greater than the second operand returns a Boolean value 1 or 0 as the output.

8.6.4 Greater than or Equal to Operator
The ">=" operator evaluates whether the first operand is greater than or equal to the second operand and returns a Boolean value 1 or 0 as the output.

8.7 Equality Operators: ==, !=

8.7.1 Equal-to Comparison Operator:
The "==" operator evaluates whether the first operand is equal to the second operand and returns a Boolean value 1 or 0 as the output.

8.7.2 Not-equal-to Comparison Operator:
The "!=" operator evaluates whether the first operand is not equal to the second operand and returns a Boolean value 1 or 0 as the output.

8.8 Bit-wise And Operator: &
The "&" operator take the binary representation of the first and second operand and does a bitwise AND operation on them.

8.9 Bit-wise Exclusive Or Operator: ^

The "^" operator take the binary representation of the first and second operand and does a bitwise Exclusive OR operation on them.

8.10 Bit-wise Or Operator: |

The "|" operator take the binary representation of the first and second operand and does a bitwise OR operation on them.

8.11 And Operator: &&

The "&&" operator returns a Boolean value of 1 if both the first and second operand evaluate to "true" else it returns a "false" or 0 Boolean value.

8.12 Or Operator: ||

The "||" operator returns a Boolean value of 1 if either the first or second operand evaluate to "true" else it returns a "false" or 0 Boolean value.

8.13 Assignment Operators: =, +=, -=, *=, /=, %=, &=, |=, ^=

8.13.1 Assignment Operator

The '=' operator assigns the expression to the right of the '=' to the identifier to the left of the '='.

8.13.2 += Operator

The "+=" operator increments the first operand by the second operand and assigns this value to the first operand.

8.13.3 -= Operator

The "-=" operator decrements the first operand by the second operand and assigns this value to the first operand.

8.13.4 *= Operator

The "*=" operator multiplies the first operand by the second operand and assigns this value to the first operand.

8.13.5 /= Operator

The "/=" operator divides the first operand by the second operand and assigns this value to the first operand.

8.13.6 %= Operator

The "%=" operator computes the remainder when the first operand is divided by the second operand and assigns this value to the first operand.

8.13.7 &= Operator

The "&=" operator computes the bit-wise and operation of the first operand and the second operand and assigns the solution to the first operand.

8.13.8 |= Operator

The "|=" operator computes the bit-wise or operation of the first operand and the second operand and assigns the solution to the first operand.

8.13.9 ^= Operator

The "^=" operator computes the bit-wise exclusive or operation of the first operand and the second operand and assigns the solution to the first operand.

9. Sample Code

The following sample code generates a photo montage of the famous *Mona Lisa* painting using numerous other images as shown on the cover page.

```
void main(String[] args){
        // Open the base image for photo montage
        Image base = Open("C:/img/mona_lisa.jpg");

        String img_dir = "C:/montage_img_lib/";

        // Create a photo montage of mona lisa
        Image result = photo_montage(base, img_dir, 3);

        // Saving the output image file
        result.saveAs("C:/img /monalisa_montage.jpg");
}

/**
 * Image base: base image for creating a photo montage
 * String img_dir: directory name where the images are stored for creating photo montage
 * int block_size: size of the smaller image blocks
 * int img_range: the number of images that will be used as the closest color; 1 means it
 *                will always use one image with the closest color to the color of an
 *                averaged block; 3 means it will select three images with the closest
 *                color and then randomly select one image to use from these three.
 */
Image photo_montage(Image base, String img_dir, int block_size, int img_range){
        // Create the target image.
        Image result = new Image(base.getWidth(), base.getHeight());
```

```
        // Go through each block of block_size of the base image to calculate the average
        // RGB value of this block, and then find an image that has the closest average
        // RGB value from the given img_dir and replace this block with this selected
        // image after resizing the width and height of this image to block_size
        for(int i = 0; I < base.getHeight(); i += block_size){
                for(int j = 0; j < base.getWidth(); j += block_size){
                        Rect block_rect = new Rect(i, j, block_size, block_size);
                        // Get a block of pixel values from the base image
                        int [][] block = base.getPixels(block_rect);

                        // Calculate the average RGB value of this block
                        Color avgRGB = calcAverageRGB(block);
                        // Get a number of images that has the closest RGB value (closest
                        // RGB distance) to this block from a image directory
                        Image [] closest_imgs = matchClosestColor(avgRGB, img_dir,
                                                                  img_range);

                        // Randomly select which closet image to use
                        // (Note: random() is a built-in function that returns a double
                        // number between 0 to 1)
                        int random = ((random()*img_range)%img_range);

                        // resize(Image i, int width, int height) is a built-in function
                        Image replace_img = resize( closest_imgs[random], block_size,
                                                    block_size); // built-in function

                        // paste(Image dest, Image src, int x_pos, int y_pos) is a built-in
                        // function and it pastes the src image to the dest image starting
                        // from left-upper corner of dest's (x_pos, y_pos) location with
                        // the width and height of src image
                        paste(result, replace_img, i, j);
                }
        }

        return result;
}
```

10. References

1. Zhou, Tiantian, Feng, Hanhua, Ra, Yong Man, Lee, Chang Woo. "Mx: A programming language for scientific computation." http://www1.cs.columbia.edu/~sedwards/classes/2003/w4115/Mx.final.pdf , May 2003.
2. Ritchie, Dennis M. *C Reference Manual*. Bell Telephone Laboratories, 1975.