Photogram

Programmable Photoshop® Language

Project Report

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

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 solves all of the existing issues on Photoshop addressed above. It will significantly save the time of people performing tedious image editions.

1.1 Goal

Photogram, a programmable Photoshop® language, is a language that enables the user to perform Photoshop operations using an iterative coding process. Photogram is meant to be easy to use, portable, powerful and expandable.

1.1.1. Ease-of-use

Photogram is a clear and intuitive language which allows users to edit photos and create animation by writing an algorithm. Like Java® or C Photogram uses a well-defined set of basic syntaxes similar to Java® and this makes Photogram programmer-friendly.

1.1.2. Portable

Java® is a cross-platform portable language due to its own interpreter, and since Photogram converts the user-created-program to Java code, it is also portable. You can execute the program on any platform where Java 1.5 Virtual Machine is installed.

1.1.3. Powerful

Photogram enables the user to edit the photo, for example sharpening, blurring, and changing colors, with just few lines of code. This power and efficiency allows work to be more productive as well as providing a clear outline of what was used to achieve the final result.

1.1.4. Expandable

Since Photogram is a programming language, it is a simple matter to add one's own functions or even import any other libraries in order to use new functions that the user needs.

1.2 Features

Our language, Photogram, supports the following features:

Syntaxes and Semantics: The syntaxes and semantics of Photogram is very similar to the Java® language with one exception for declaring macros, in which case C++ syntax is used.

Functions: Photogram supports the implementation of functions with a return type and parameters. The syntax for writing a function is the same as the Java® language.

Importing: Photogram supports importing functions written in other .pg (Photogram extension) files by using "#include" statement.

Non-Object Oriented: Photogram does NOT support implementation of classes/objects; however, Photogram provides several built-in classes such as Image, Font, Color, Pixel, Line, Rectangle, and Oval.

Arrays: Photogram supports arrays, the syntax for declaring an array being the same as that used in the Java® language.

Editor: We provide a specialized editor for Photogram, which will open new picture-viewable windows so that users can easily find out the coordinates of the desired points in the input images. However, a programmer can choose not to use our editor and use any other word editors.

Image Processing: Photogram provides some built-in image processing functions that are useful or difficult to implement using Photogram code.

Caption: Photogram supports the ability to create captions, drawing a string of text on the image.

2 Language Tutorial

2.1 How to use Photogram

2.1.1 How to compiler Photogram program

To compile a photogram program use the PGCompiler, and type the following command.

```
C:\java PGCompiler PGSampleProgram.pg
```

After using PGCompiler to compiler the photogram code, and there after generating the java code, the user needs to compile the java code using the java compiler.

```
C:\javac PGSampleProgram.java
```

2.1.2 How to run Photogram program

To run a photogram program use the following command.

```
C:\java PGSampleProgram
```

2.2 Image Processing Applications

Using the library functions provided in photogram we can perform many different image processing routines, like gray-scaling, low-pass filtering, high-pass filtering, edge detection, binarization, rotation, to name a few. The results obtained have been show in Fig 2.1.

```
PGImageProcessing.pg
void main(String args[]) {

    // Opens images from a directory
    Image src = new Image();
    Image output = new Image();
    src.open("columbia.jpg", Image.RGB_TYPE);
    int w = src.getWidth();
    int h = src.getHeight();
    Image outputFile = new Image(3 * w, 3 * h);

    output = ditherBright(src);
    pasteImage(outputFile, output, 0, 0);

    output = filterGaussian(src);
    pasteImage(outputFile, output, w, 0);

    output = quantPopulosity(src);
    pasteImage(outputFile, output, 2 * w, 0);

    output = grayscale(src);
    pasteImage(outputFile, output, 0, h);
```

```
        output = rotate(src, 90.0);
        pasteImage(outputFile, output, w, h);

        output = filterBartlett(src);
        pasteImage(outputFile, output, 2 * w, h);

        output = filterBox(src);
        pasteImage(outputFile, output, 0, 2 * h);

        output = filterEnhance(src);
        pasteImage(outputFile, output, w, 2 * h);

        output = filterEdgeDetect(output);
        pasteImage(outputFile, output, 2 * w, 2 * h);

        outputFile.saveAs("processed_images.jpg", Image.JPEG);
    }

    Image pasteImage(Image dest, Image src, int x, int y) {

        for (int i = 0; i < src.getWidth(); i++) {
            for (int j = 0; j < src.getHeight(); j++) {
                Color color = new Color();
                color = src.getColor(i, j);
                Pixel pix = new Pixel(color, i + x, j + y);
                dest.setPixel(pix);
            }
        }
        return dest;
    }
```

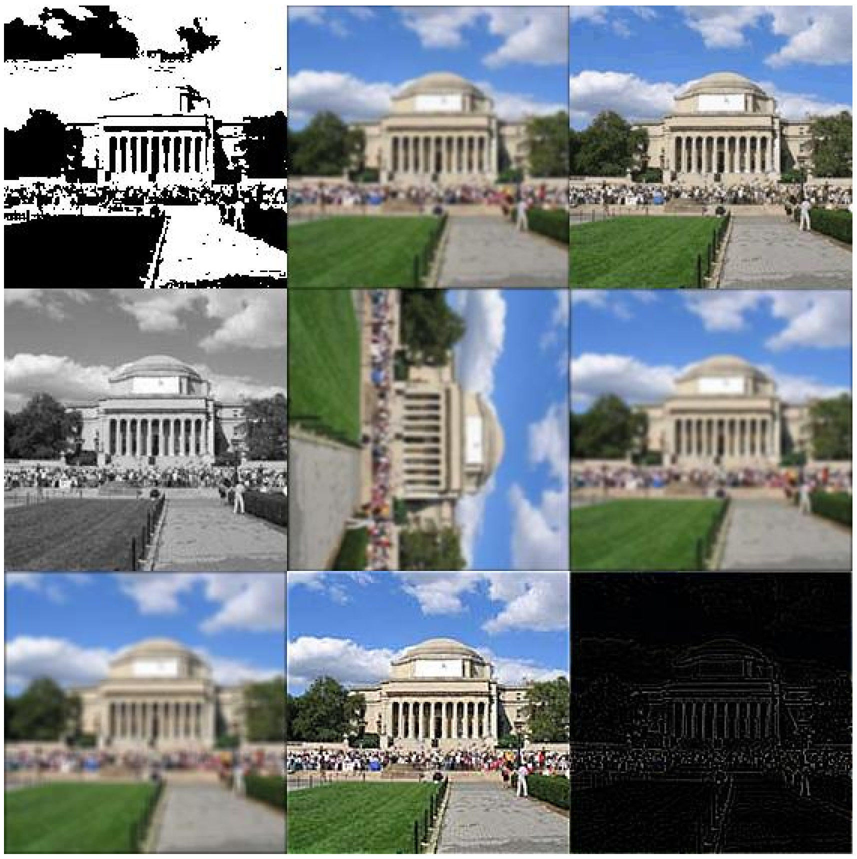


Fig 2.1 Using Built-in image processing techniques provided in Photogram library

2.3 Image Manipulation: Collage

In the following example, we create a collage of three input images, with captions. In this example, the user can run the code, see the collage output, and go back and change the coordinated in the program and run the code again. So, no history of previous actions needs to be saved as .psd file. Also, we have shown how "**#include PGSampleImport.pg"** can be used to import function and globals from other Photogram files. The collage created is shown in Fig 2.2.

`PGCollage.pg`

```
#include "PGCreateCollage.pg"

void main(String args[]) {
    Image myImages[] = new Image[3];
    for (int i = 0; i < 3; i++) {
        myImages[i] = new Image();
    }

    myImages[0].open("collage01.jpg", Image.RGB_TYPE);
    myImages[1].open("collage02.jpg", Image.RGB_TYPE);
    myImages[2].open("collage03.jpg", Image.RGB_TYPE);

    String captions[] = new String[3];
    captions[0] = "Column";
    captions[1] = "Escalator";
    captions[2] = "Library";

    // Calling the function collage
    Image result = new Image();
    result = createCollage(myImages, captions, 550, 400, 3, 1);

    // Saving the output image file
    result.saveAs("tcollage.jpg", Image.JPEG);
}
```

`PGCreateCollage.pg`

```
#include "PGPasteImage.pg"

Image createCollage(Image images[], String captions[], int width, int
height, int row, int col) {

    // Creating the target image.
    Image result = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Color color = new Color(255, 255, 255);
            Pixel pix = new Pixel(color, i, j);
            result.setPixel(pix);
        }
    }

    // Setting the font to be used for adding text onto the input images
    Font font = new Font(Font.ARIAL, 15, 1);
    Color color = new Color(0, 0, 0);

    result = pasteImage(result, images[0], 10,
                        400 - images[0].getHeight() - 50);
    result = drawText(result, captions[0], 50, 380, font, color);
    result = pasteImage(result, images[1], 150,
                        400 - images[1].getHeight() - 50);
    result = drawText(result, captions[1], 190, 380, font, color);
    result = pasteImage(result, images[2], 330,
                        400 - images[2].getHeight() - 50);
    result = drawText(result, captions[2], 400, 380, font, color);

    return result;
}
```

PGPasteImage.pg

```
Image pasteImage(Image dest, Image src, int x, int y) {

    for (int i = 0; i < src.getWidth(); i++) {
        for (int j = 0; j < src.getHeight(); j++) {
            Color color = new Color();
            color = src.getColor(i, j);
            Pixel pix = new Pixel(color, i + x, j + y);
            dest.setPixel(pix);
        }
    }
    return dest;
}
```

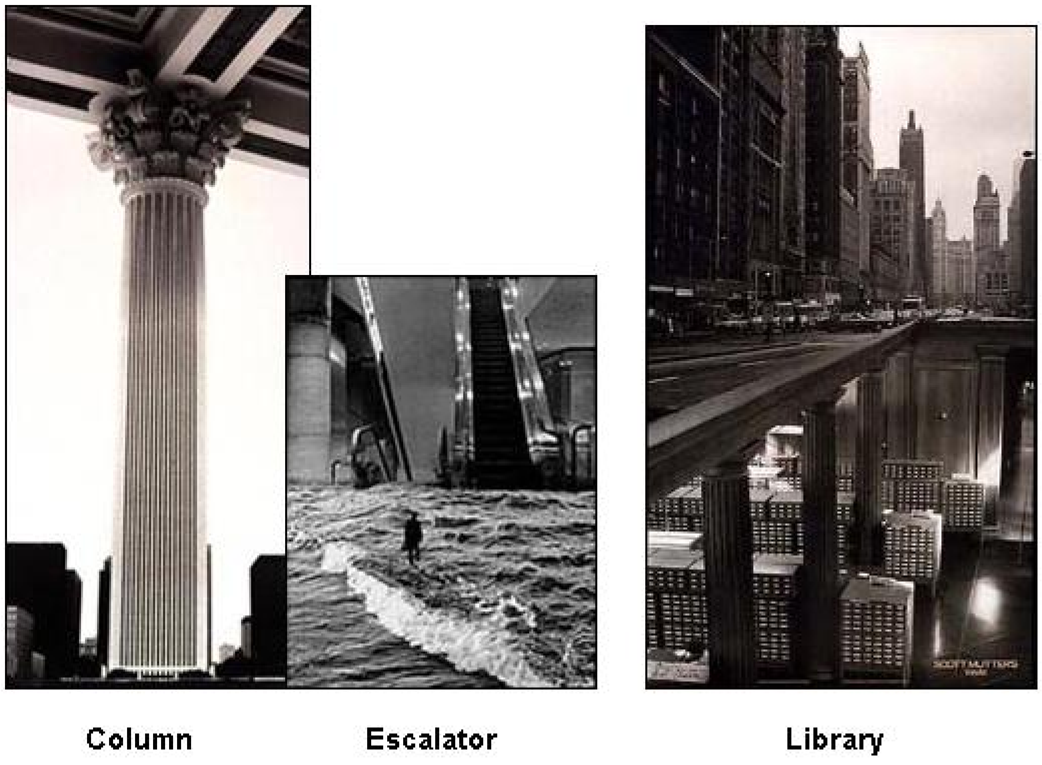


Fig 2.2 Collage

2.4 Computer Vision Applications: Optical Flow

Optical flow is a concept for considering the motion of objects within a visual representation. It is closely related to motion estimation. The program **PGOpticalFlow.pg** computes optical flow between two gray-level consecutive images in a video, and produces a gray-level output image, showing the optical flow vector field as a "needle map" of a given resolution. The needle map is drawn on a 16x16 grid (i.e. the optical flow vector field sampled every 16 pixels along x and y axes). From each grid point a line is drawn in the direction of the optical flow vector with a length proportional to the vector's magnitude. Since, Photogram supports, image manipulation, line and circle drawing functions, a basic optical flow algorithm can be implemented very easily. The result obtained is show in Fig 2.3.

```
int gridSize = 16;

void main(String args[]) {

    Image inputImageOne = new Image();
    Image inputImageTwo = new Image();
    Image outputImage = new Image();

    inputImageOne.open("flow1.jpg", Image.RGB_TYPE);
    inputImageTwo.open("flow2.jpg", Image.RGB_TYPE);
    outputImage.open("flow1.jpg", Image.RGB_TYPE);

    outputImage = computeFlow(inputImageOne, inputImageTwo, outputImage,
                              gridSize);
    outputImage.saveAs("opticalFlow.jpg", Image.JPEG);
}

Image computeFlow(Image imageOne, Image img2, Image img3, int step) {
    int rows = imageOne.getHeight();
    int columns = imageOne.getWidth();
    double correlation = 0.0;
    double patchVar = 0.0;
    double u, v;
    int k = 15;
    int qx = step, qy = step;
    double newC;
    double mean;
    double scale = 0.0;

    for (int j = 15; j < columns - 15; j++) {
        qy = 0;
        if (qx == step) {
            for (int i = 15; i < rows - 15; i++) {
                if (qy == step) {
                    correlation = 0.0;
                    u = 0.0;
                    v = 0.0;
                    for (int n = -k; n < k; n++) {
                        for (int m = -k; m < k; m++) {
                            /* Do Template Matching */
                            newC = match(imageOne, img2, i, j, i+ m, j + n);
                            if (newC > correlation) {
                                /* Compute (u,v) */
                                correlation = newC;
                                u = m;
                                v = n;
                            }
                        }
                    }
                    mean = computeMean(imageOne, i, j);
                    patchVar = computeVar(imageOne, i, j);
                    if (patchVar > 12.0 && mean > 15.0 &&
                        correlation > 0.995) {
                        u /= 12;
                        v /= 12;
                        Line line = new Line(j, i, j+doubleToInteger(10* v),
                                             i + doubleToInteger(10 * u));
                        Oval oval = new Oval(j, i, 2, 2);
                        Color color = new Color(255, 255, 255);
                        img3 = drawShape(img3, line, color);
                        img3 = drawShape(img3, oval, color, true);
                    }

                    qy = 0;
                    qx = 0;
                }
                /* 2nd mod */
                qy++;
            }
        }
        /* 1st mod */
        qx++;
    }

    return img3;
}

double match(Image imageOne, Image imageTwo, int index_a, int index_b,
             int index_c, int index_d) {

    double cor = 0.0, sum1 = 0.0, sum2 = 0.0, sum3 = 0.0;

    for (int i = -3; i < 3; i++) {
        for (int j = -3; j < 3; j++) {
            Color colorOne = new Color();
            Color colorTwo = new Color();
            colorOne = imageOne.getColor(index_b + j, index_a + i);
            colorTwo = imageTwo.getColor(index_d + j, index_c + i);
            cor += (colorOne.g * colorTwo.g);
            sum1 += (colorOne.g * colorOne.g);
            sum2 += (colorTwo.g * colorTwo.g);
        }
    }

    sum1 = sqrt(sum1);
    sum2 = sqrt(sum2);
    cor /= sum1;
    cor /= sum2;

    return cor;
}

double computeVar(Image imageOne, int m, int n) {

    double var = 0.0;
    double mean = 0.0;

    for (int i = -3; i < 3; i++) {
        for (int j = -3; j < 3; j++) {
            Color color = new Color();
            color = imageOne.getColor(n + j, m + i);
            mean += color.g;
        }
    }
```

```
        mean /= 49.0;

        for (int i = -3; i < 3; i++) {
            for (int j = -3; j < 3; j++) {
                Color color = new Color();
                color = imageOne.getColor(n + j, m + i);
                var += pow(color.g - mean, 2.0);
            }
        }

        var /= 49.0;
        var = sqrt(var);

        return var;
    }

    double computeMean(Image imageOne, int m, int n) {

        double mean = 0.0;

        for (int i = -3; i < 3; i++) {
            for (int j = -3; j < 3; j++) {
                Color color = new Color();
                color = imageOne.getColor(n + j, m + i);
                mean += color.g;
            }
        }
        mean /= 49.0;
        return mean;
    }
```

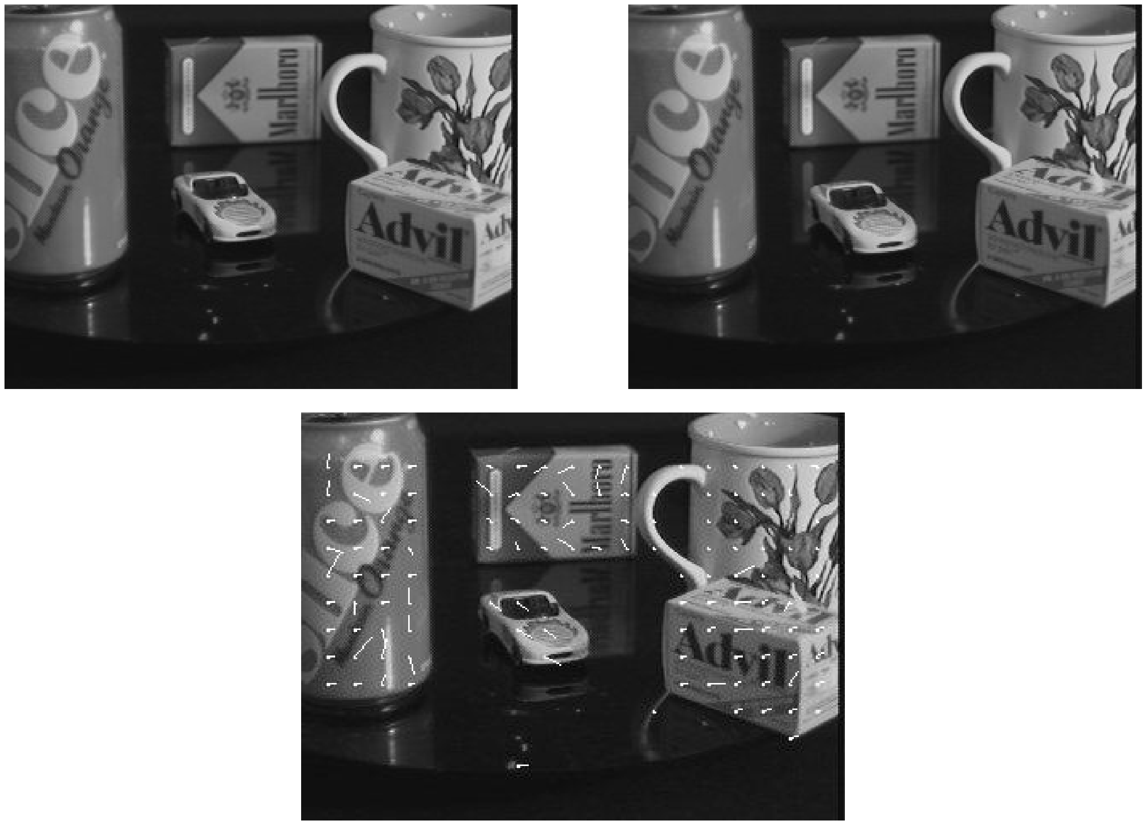


Fig 2.3: Optical Flow

2.5 Image Manipulation: Photomontage

Photomontage is an art composition created by arranging multiple photographs into one; often, uses photographs that hold elements or represent a single theme that the artist is trying to express. To create a photomontage of any given image, one would probably need to buy software which has its own image database.

Using Photogram one can create photomontage of any image, whether color or black and white in as less as 200 lines of Photogram code. First, we need to create an image database, wherein one has the option of using their personal photographs. Secondly, we need to set a few parameters in the program **PGPhotomontage.pg**, namely the number of images in the database, location of the database, the source image to be manipulated, and finally setting the size of the sources image patches which will be replaced by the images in the database.

The algorithm which we have used, first matches the average RGB value of the source image patch with the average RGB value of the image in the database. If they differ by a value less than the predetermined threshold, then we proceed to correlate the two images, and if their correlation percentage is more than the predetermined threshold, we select the image. The above process is performed iteratively till we obtain the best match for the image patch.

Also, we desire that we should have as many difference images as possible for same average RGB value. So, we have ensured that while creating the Photomontage of any image, the program doesn't reuse images from the database as far as possible. The computation time increases as we increase the images in the database, and also with decreasing the size of the source image patch to be replaced (as we will be using more images for creating the photomontage).

We have tested our algorithm implemented in Photogram, with varying image patch sizes of 16x16, 24x24 and 32x32, on Prof. Stephen Edwards face, and the output is shown in Fig 2.4. The full sized photomontage images can be seen in the Appendix.

PGPhotomontage.pg

```
int totalImageNum = 2008;
int size = 32;
boolean imageUsed[] = new boolean[totalImageNum];

void main(String args[]) {
    // Open the base image for photo montage
    Image base = new Image();
    base.open("c-edwards.jpg", Image.RGB_TYPE);
    String img_dir =
          "C:/Documents and Settings/Neesha/My
   Documents/Columbia/PLT/PhotoGram/Photogram/Images/";

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

    // Saving the output image file
    result.saveAs("PG_output.jpg", Image.JPEG);
}

/**
 * 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
 */
Image photo_montage(Image base, String img_dir, int block_size_x,
                    int block_size_y) {
    // Create the target image.
    Image result = new Image(base.getWidth(), base.getHeight());
    for (int i = 0; i < totalImageNum; i++) {
        imageUsed[i] = false;
    }
    // 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.getWidth(); i += block_size_x) {
        for (int j = 0; j < base.getHeight(); j += block_size_y) {

            Image block = new Image(block_size_x, block_size_y);
            for (int m = 0; m < block_size_x; m++) {
                for (int n = 0; n < block_size_y; n++) {
                    Color color = base.getColor(i + m, n + j);
                    Pixel pix = new Pixel(color, m, n);
                    block.setPixel(pix);
                }
            }

            // 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_img = matchClosestColor(base, avgRGB, img_dir,
                                                  i, j);

            // Randomly select which closet image to use

            for (int m = 0; m < block_size_x; m++) {
                for (int n = 0; n < block_size_y; n++) {
                    Color color = closest_img.getColor(m, n);
                    Pixel pix = new Pixel(color, m + i, n + j);
                    result.setPixel(pix);
                }
            }
        }
    }
    return result;
}

Color calcAverageRGB(Image image) {
    Color color = new Color(0, 0, 0);
    int colorNum = image.getWidth() * image.getHeight();

    for (int x = 0; x < image.getWidth(); x++) {
        for (int y = 0; y < image.getHeight(); y++) {
            Color imgC = new Color();
            imgC = image.getColor(x, y);
            color.r += imgC.r;
            color.g += imgC.g;
            color.b += imgC.b;
        }
    }

    color.r /= colorNum;
    color.g /= colorNum;
    color.b /= colorNum;

    return color;
}

Image matchClosestColor(Image srcImage, Color color, String img_dir,
                        int index_a, int index_b) {
    Image images = new Image();
    Image allImages[] = new Image[totalImageNum];
    allImages[0] = new Image();
    allImages[0].open(img_dir + "result0.jpg", Image.RGB_TYPE);
    images.copy(allImages[0]);

    double colorDiff = 0;
    double minDiff = 60.;
    int minIndex = 0;
    int count = 0;
    double maxCor = 0.75;
    boolean breakOut = false;

    for (int i = 0; i < allImages.length; i++) {
        if (imageUsed[i] == true) {
            continue;
        }

        if (breakOut == true) {
            break;
        }
        allImages[i] = new Image();
        allImages[i].open(img_dir + "result" + i + ".jpg", Image.RGB_TYPE);
        Color avgRGB = new Color();
        avgRGB = calcAverageRGB(allImages[i]);
        colorDiff = sqrt(pow((color.r - avgRGB.r), 2) +
                         pow((color.g - avgRGB.g), 2) +
                         pow((color.b - avgRGB.b), 2));
        if (colorDiff < minDiff) {
            double cor = match(srcImage, allImages[i], index_a, index_b);
            if (cor >= maxCor) {
                images.copy(allImages[i]);
                minDiff = colorDiff;
                minIndex = i;
                maxCor = cor;
                if (maxCor > 0.95 && colorDiff < 6) {
                    breakOut = true;
                }
                count++;
            }
        }
    }

    if (count > 0) {
        imageUsed[minIndex] = true;
    }

    if (count == 0) {
        minDiff = 30.;
        minIndex = 0;
        maxCor = 0.75;
        count = 0;
        breakOut = false;

        for (int i = 0; i < allImages.length; i++) {
            if (imageUsed[i] == false) {
                continue;
            }
            if (breakOut == true) {
                break;
            }
            allImages[i] = new Image();
            allImages[i].open(img_dir + "result" +i+".jpg", Image.RGB_TYPE);
            Color avgRGB = new Color();
            avgRGB = calcAverageRGB(allImages[i]);
            colorDiff = sqrt(pow((color.r - avgRGB.r), 2) +
                             pow((color.g - avgRGB.g), 2) +
                             pow((color.b - avgRGB.b), 2));
            if (colorDiff < minDiff) {
                double cor = match(srcImage,allImages[i], index_a, index_b);
                if (cor >= maxCor) {
                    images.copy(allImages[i]);
                    minDiff = colorDiff;
                    minIndex = i;
                    maxCor = cor;
                    if (maxCor > 0.95 && colorDiff < 6) {
                        breakOut = true;
                    }
                    count++;
                }
            }
        }

    }

    return images;
}

double match(Image imageOne, Image imageTwo, int index_a, int index_b) {

    double cor = 0.0, sum1 = 0.0, sum2 = 0.0, sum3 = 0.0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            Color colorOne = new Color();
            Color colorTwo = new Color();
            colorOne = imageOne.getColor(index_a + i, index_b + j);
            colorTwo = imageTwo.getColor(i, j);
            cor += (colorOne.r * colorTwo.r);
            cor += (colorOne.g * colorTwo.g);
            cor += (colorOne.b * colorTwo.b);
            sum1 += (colorOne.r * colorOne.r);
            sum1 += (colorOne.g * colorOne.g);
            sum1 += (colorOne.b * colorOne.b);
            sum2 += (colorTwo.r * colorTwo.r);
            sum2 += (colorTwo.g * colorTwo.g);
            sum2 += (colorTwo.b * colorTwo.b);

        }
    }

    sum1 = sqrt(sum1);
    sum2 = sqrt(sum2);
    cor /= sum1;
    cor /= sum2;

    return cor;
}
```

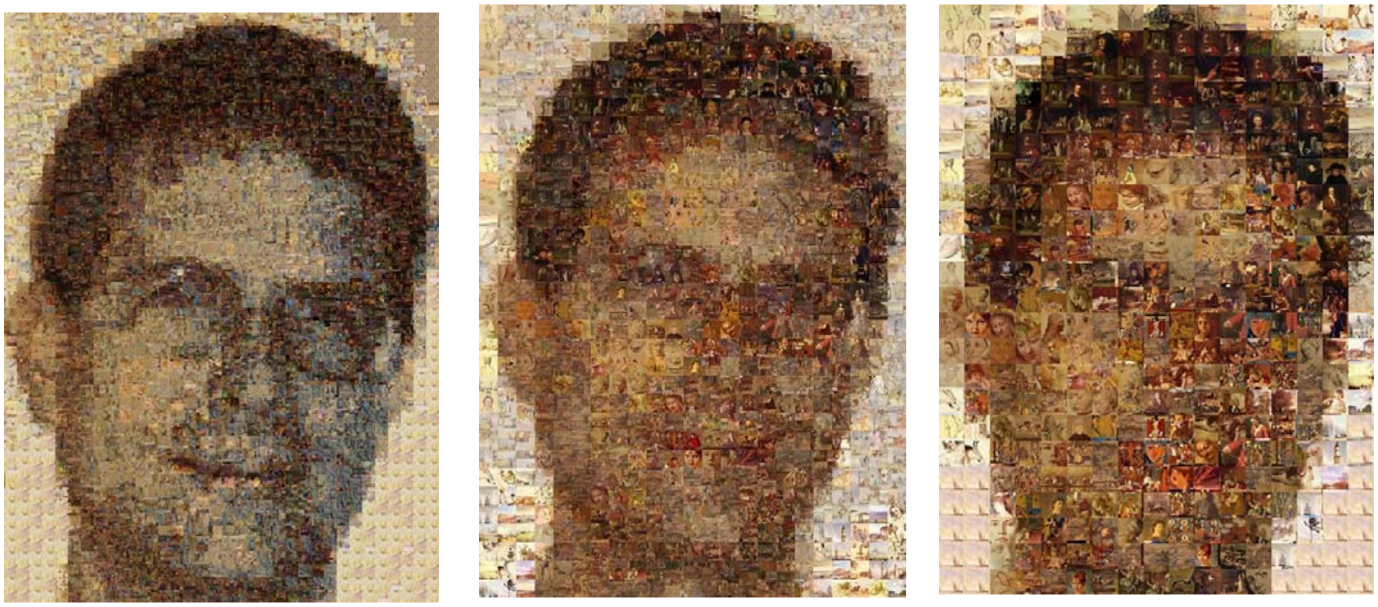


Fig 2.4: Photomontage algorithm tested on Prof. Stephen Edwards face

3 Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

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

3.1.1.1 Multiple Line Comment

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

3.1.1.2 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 */
```

3.1.2 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.

3.1.3 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, else, while, switch, case, for, true, false, continue, break, void, return, null, new

3.1.4 Constants

There are three kinds of constants as follows.

3.1.4.1 Integer Constant

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

3.1.4.2 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.

3.1.4.3 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."
```

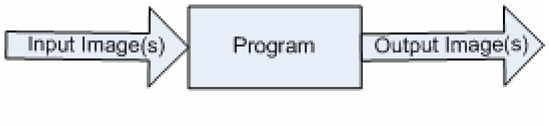
3.1.5 Other Tokens

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	.

3.2 Input and Output of Photogram

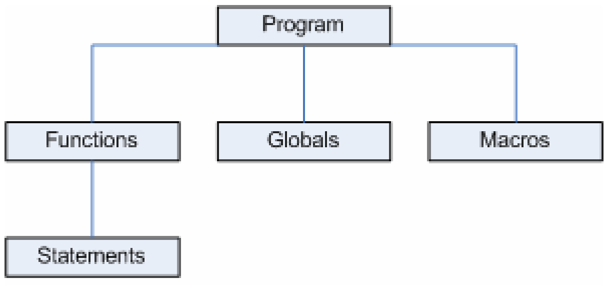
Given a Photogram program, the input into the program are images which are specified as an argument before running the program and the output of the program will be also images that have been modified or created by the program as shown in the figure below.



Inputs and Outputs of a Program

3.3 Program Structure

A Photogram program consists of three components, functions, globals and macros. 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

3.3.1 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.

3.3.1.1 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.

3.3.1.1.1 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;
...
}

3.3.1.1.2 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>
}

3.3.1.1.3 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;

3.3.1.1.4 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;

3.3.1.1.5 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>);

3.3.1.1.6 Object Call Statement

This statement is used to call a member of the built-in-objects. The object call statement is used in the following manner.

<l-value>.<member-of-object>;

3.3.1.1.7 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>;

3.3.1.1.8 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>;

3.3.1.1.9 Primitive Declaration Statement

This statement shows how to declare a primitive:
<primitive> <identifier>;

3.3.1.1.10 Built-in Object Declaration Statement

This statement shows how to declare a built-in object:
<built-in object> <identifier> = new <built-inobject>(<expression list>);

3.3.1.1.11 Array Declaration Statement

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

3.3.2 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.

3.3.3 Macros

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

3.3.3.1 Include Macros

#include <fine name>

3.3.3.2 Define Macros

#define <identifier> <constant>

3.4 Scope

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

3.5 Namespace

3.5.1 Function namespace

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

3.5.2 Variable namespace

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

3.6 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.

3.6.1 Primitives

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

3.6.1.1 int

32-bit integer

3.6.1.2 double

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

3.6.1.3 boolean

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

3.6.1.4 void

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

3.6.2 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)

3.6.2.1 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

3.6.2.2 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);

3.6.2.3 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

3.6.2.4 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

3.6.2.5 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

3.6.2.6 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

3.6.2.7 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);

3.6.2.8 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);

3.7 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.

3.7.1 Parentheses

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

3.7.2 Object call: .

The object call operator is used to call the member of the built-in-objects.

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

3.7.3.1 Not Operator

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

3.7.3.2 Increment Operator

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

3.7.3.3 Decrement Operator

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

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

3.7.4.1 Multiplication Operator (*)

The operator ‘*’ multiplies the two numeric tokens.

3.7.4.2 Division Operator (/)

The operator ‘/’ divides the first operand with the second operand.

3.7.4.3 Modulo Operator (%)

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

3.7.5 Addition and Subtraction Operators: +, -

3.7.5.1 Addition Operator

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

3.7.5.2 Subtraction Operator

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

3.7.6 Shift Operators: >>, <<

3.7.6.1 Left Shift Operator

The “<<” operator shifts the bits of the first operand left by the number of bits specified by the second operand.

3.7.6.2 Right Shift Operator

The “>>” operator shifts the bits of the first operand right by the number of bits specified by the second operand.

3.7.7 Relation Operators: <, <=, >=, >

3.7.7.1 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.

3.7.7.2 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.

3.7.7.3 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.

3.7.7.4 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.

3.7.8 Equality Operators: ==, !=

3.7.8.1 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.

3.7.8.2 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.

3.7.9 Bit-wise And Operator: &

The “&” operator take the binary representation of the first and second operand and does a bitwise AND operation on them.

3.7.10 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.

3.7.11 Bit-wise Or Operator: |

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

3.7.12 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.

3.7.13 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.

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

3.7.14.1 Assignment Operator

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

3.7.14.2 += Operator

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

3.7.14.3 -= Operator

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

3.7.14.4 *= Operator

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

3.7.14.5 /= Operator

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

3.7.14.6 %= Operator

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

3.7.14.7 &= 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.

3.7.14.8 |= 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.

3.7.14.9 ^= 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.

4 Project Plan

4.1 Process used for planning, specification, development and testing:

We had weekly meeting at one of our team member’s place on Thursday and Friday for about 4~5 hours each day. For each meeting, the leader was responsible for bringing up the agenda for the meeting. The leader inquires every team member’s progress, and then decides the goal and plan for the following week. We used CVS for sharing our codes, and MSN messenger for communicating and reporting problems.

We brainstormed the big picture of our project specification together, and as we move on through the semester, we changed some of our specifications due to difficulties of implementing certain specifications.
We developed our codes together at the time of weekly meeting and discussed specification changes. Besides these 4~5 hours meeting on Thursday and Friday, we individually worked on each of our designated part on the other days of the week.

We tested each small part of our codes by writing a simple Photogram code, and checked whether the output is same as the expected one. Whenever there was a bug, we used JBuilder's powerful debugger to step through the codes except for ANTLR codes. We have designated one of our team members to write many codes written in Photogram and report bugs (if any) to the rest of the members through email.

4.2 Programming style guide used by the team

Java Coding Style:

At the beginning of the semester, all of our team members have agreed to use JBuilder, which is a Java editor software developed by Borland, for writing, debugging, and running Java codes, and we used the following styles.

- Use prefix "PG" for each class we create except for those that are only used by our library class (PGLibrary)
- At the beginning of each java file, include a description and the author information (the format is automatically generated by JBuilder)
- At the beginning of each method, include a brief description of what the function does in the format that JBuilder accepts for automatically generating HTML file from those comments
    - Example:
    - /**
    - * Get the current enviroment variables
    - * @return PGEnvironment
    - */
    - public PGEnvironment getEnv(){
    -     return env;
    - }
- Use proper indentation every time entering a new scope using tab keys, and each tab key space is 4 spaces (JBuilder is set to use 4 spaces as 1 tab space, so every time it enters a new scope, JBuilder automatically indents)
- Codes in the same scope should have same number of indentations (taken care of by JBuilder)
- For comments other than describing the classes, functions, and member fields, use "// comments" instead of "/* comments */"

- Name of constants declared by using “final” should have all capital letters (i.e. CONSTANT)
- Variables’ names should start with non-capital letter

4.3 Project timeline

Date	**Description**
Sep. 27th	Proposal Due
1st week of Oct.	Start Lexer / Parser
2nd week of Oct.	Debug Lexer / Parser
Oct. 20th	Language Reference Manual Due
4th week of Oct.	Finalize Lexer / Parse, Start Walker
1st week of Nov.	Start Semantic Analyzer
2nd week of Nov.	Implement library functions
3rd & 4th week of Nov.	Debug Walker / Semantic Analyzer
1st week of Dec.	Finalize Walker / Semantic Analyzer
2nd week of Dec.	Start extensive testing
3rd week of Dec.	Documentation / Make sample programs
Dec. 20th	Final Report Due / Presentation

4.4 Roles and responsibilities of each team member

Name	**Responsibilities**
Ohan Oda (Leader)	Semantic Analyzer, Library, Editor
Neesha Subramaniam	Parser, Sample programs, Documentation
Richard Ng	Parser, Walker
Seikwon Kim	Testing, Documentation

4.5 Software development environment used (tools and languages)

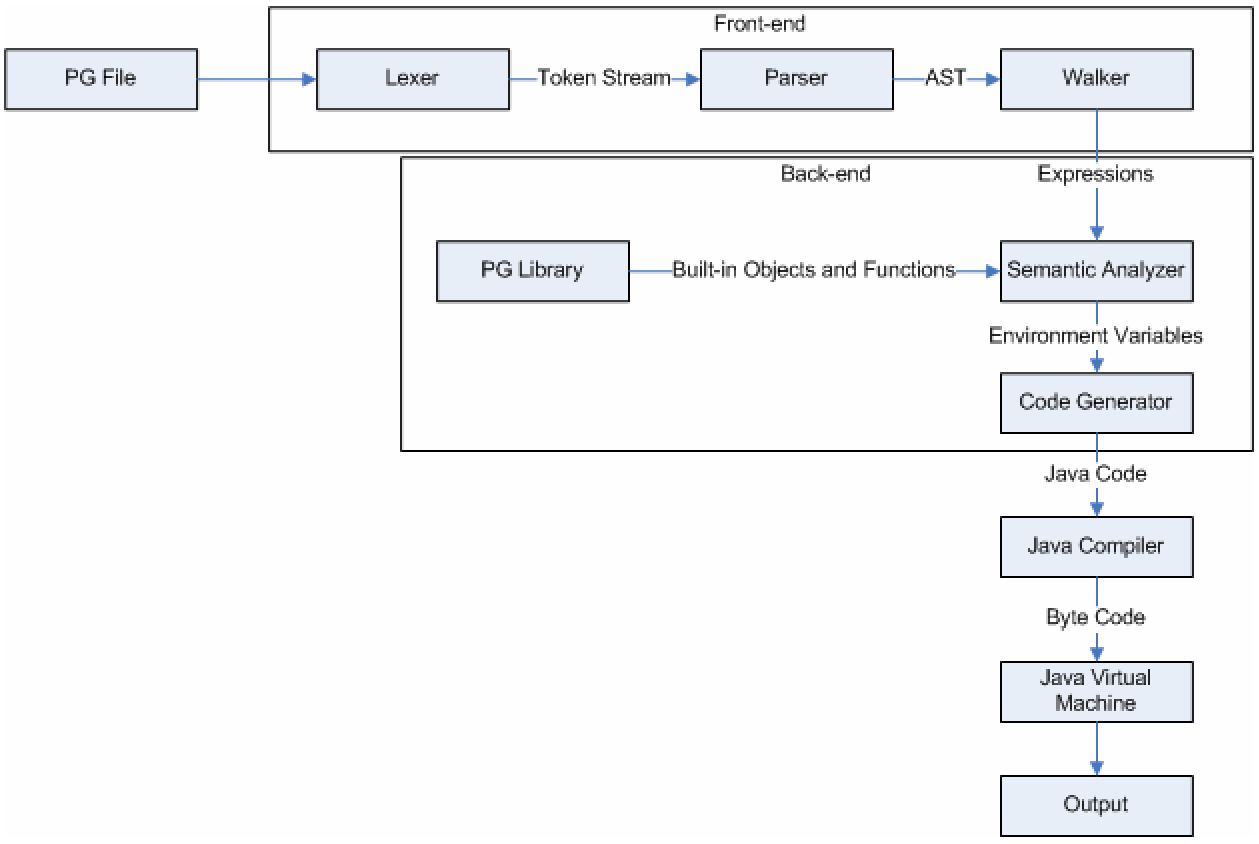
Most of the programs are written in Java under JDK 1.5.0 environment using JBuilder 2005 Personal Edition. JBuilder has user friendly graphic user interface similar to other Java editors like Eclipse and JCreator. The difference is that it has much more powerful debugger, and it can automatically generate Javadoc in HTML format from appropriately formatted comments. It also does automatic indentation and import optimization. Lexer, parser, and walker were coded in Antlr, and we used the generated Java programs. We developed the programs under Window XP Operating Systems. We used CVS to exchange and update files, and used MSN messenger to communicate each other.

4.6 Project Log

Date	**Notes**
9 Sep 2005	Brainstorming session for project specifications
16 Sep 2005	Birth of Photogram.
23 Sep 2005	White paper finalized
30 Sep 2005	Grammar file started.
7 Oct 2005	WinCVS setup.
14 Oct 2005	Lexer and Parser version one finished
20 Oct 2005	Language Reference Manual finalized.
27 Oct 2005	Library Implementation started
3 Nov 2005	Library implementation completed
10 Nov 2005	Parser version two finished
17 Nov 2005	Walker and Semantic Analyzer started.
1 Dec 2005	Work on sample programs and complete testing for semantics started
14 Dec 2005	Project Report work started.
20 Dec 2005	Photogram ready for Market Release.

5 Architectural Design

5.1 Block diagram showing the major components of our translator



5.2 Interfaces between the components

There are essentially only two components to the Photogram architecture, the front-end and the back-end.

5.2.1 Front-end

In the front-end, the only source of input that Photogram accepts is a file with extension “pg.” This front-end is done through a single file named PGCompiler. PGCompiler takes as an argument a file name and passes the file first to the lexer. The lexer goes through the file and passes a stream of tokens to the parser. The parser constructs an Abstract Syntax Tree (AST) from the stream of tokens and passes this to the walker. If there any syntactical errors discovered during the conversion to the AST tree, the walker is not executed and only the syntax errors are reported to the programmer. The AST tree is passed to two walkers, the first walker is known as the prewalker and only deals with function declarations as well as global variable definitions and includes. The second walker is simply known as the walker and makes sure that all the statements in the functions are correct.

5.2.2 Back-end

In the back-end the semantic analyzer takes care of all semantic error analysis for everything. The same semantic analyzer is used for the prewalker and the regular walker. When an instance of the semantic analyzer is created all the built-in objects and library functions are loaded as well as initial set up of the symbol table. The semantic analyzer does not do any interpreting and only checks for semantic errors present. If there are no semantic errors present, then the code generator is invoked and the pg file is converted into java. This java file is converted into byte code through the java compiler and then run on the java virtual machine resulting in the desired output.

5.3 Work Subdivision

Lexer	Richard Ng, Neesha Subramaniam
Parser	Richard Ng, Neesha Subramaniam
Walker	Richard Ng
Semantic Analyzer	Ohan Oda
PG Library	Ohan Oda
Code Generator	Ohan Oda

6 Test Plan

6.1 Goal

Since our language, Photogram, is a java-like language which provides variety of manipulative functions, testing all the functions and grammars was necessary. The major goal of testing was to make as many bugs as possible to realize the problem in our language; thus to fix the language correctly.

6.2 Phases

Test phases involved not only at the final stage of development but also involved at the initial stage of language development. There were basically three test phases that the testing was done.

6.2.1 Phase I: Testing Grammar

The major feature of test code in phase I was to check whether grammar parses everything correctly. The test suit is therefore one long file which everything should perfectly be parsed to AST tree.

6.2.2 Phase II: Connectivity between Parser and Semantic Analyzer

This phase occurred when grammar, walker and semantic were more than half done. By this time, there were still some grammar and walker problems. The major feature of the test codes was to check whether parser parses grammar to semantic analyzer correctly. The test suit is composed of 21 test files, each file testing functions, operators, built-in-objects, etc. Every time an update was made in the grammar, walker or semantic analyzer code, these tests were run to check whether the updates were successful or not. These test files were quite successful in determining whether a source code update caused any new bugs.

6.2.3 Phase III: Final Testing

In this phase, we tested the perfectness of the language. The test codes were not module-by-module, but test codes were actually sample codes.

6.3 Sample Test Codes

6.3.1 Grammar Test Code

This code was to test whether parser correctly generates AST or not. This is lexically not a correct grammar.

```
#define TEST 5
  #include "library.pg"

  int Global_int;
  int Global_intTest = 5;
  int array[2][3][5];

  String Global_str;
  String Global_str = "String Testing";
  String Global_strTest = new String("New String Test");

  double Global_Dbl;
  double Global_DblTest = 2.34e+2;

  boolean Global_bool;
  boolean Global_boolTest = true;

  Line Global_line;
  Line Global_lineTest = new Line(15, Global_int *= 4, Global_intTest,
  Global_intTest);

  Color Global_color;
  Color Global_colorTest = new Color(Global_int = 3+2, 20,
  Global_intTest, Global_intTest);

  Pixel Global_pixel;
  Pixel pixel2 = new Pixel(Global_colorTest, Global_intTest, 30);

  Rect Global_rect;
  Rect Global_rectTest = new Rect(Global_intTest = 2, Global_intTest,
  Global_int = 5, Global_intTest);

  Oval Global_oval;
  Oval Global_ovalTest = new Oval(Global_int, 29, Global_intTest,
  Global_int = 23);
```

```
int Num_Func() //Functions to Test Local & Global Numbers
{
     int Local_int;
     double Local_dbl;

     Local_int = 5;
     Local_dbl = 23.1;

     Global_DblTest = 24.23e-23 - 2 * 1;
     Global_intTest = 10;

     Global_int = Call_Num_Func(Local_int *= Global_intTest);
Global_int = Call_Num_Func(Global_intTest++);
     Global_intTest = Local_int--; //Global_intTest = 4; Local_int = 3
     Local_int = Recall_Num_Func(30);

     return Global_intTest;
}

int Recall_Num_Func(int TestNum) //Functions to Test Global_int Returns
{
     return TestNum;
}

void Image_Func()
{
     int height = 10;

     Image img[2];
     Image img2 = new Image(1024, 10+5);

     Font font;
     Font font2 = new Font(Font.ARIAL, 12, Font.BOLD);

     img[0].getImage();
     img[0].saveAs("Testing",img[0].JPEG_FORMAT);

     img[1].getImage();
     img[1].saveAs("Test",img[1].JPEG_FORMAT);

     Global_rect = new Rect(Global_int *= 2 + Global_intTest,
Global_intTest = Global_int%Global_intTest, Global_int +=
Global_intTest + 3, Global_intTest -= Global_int++);
     //Test for Many Number Operators that Should work!!!!!
}

String Method_Func(boolean TestBool)
{
     String LocalString = new String("Hello Richard, Hello Neesha");
     TestBool = true;
     int TestSwitch = 2;

     if(TestBool)
     {
            switch(TestSwitch)
            {
                   case 1:
                          for(int i = 0; i<= 20; i = i+3+1)
                          {

     drawShape(Global_lineTest,Global_colorTest);
                          }
                          break;
```

```
                  case 2:
                        if(TestSwitch != TestSwitch)
                              continue;
                        else
                              break;
            }
      }
      else if(!TestBool)
      {
            drawShape(Global_oval,Global_colorTest);
            while(TestBool == false)
                  TestBool = !!!!!(!!TestBool); //TestBool goes TRUE
and comes out
      }
      return LocalString;
}

int Bitop_Func()
{
      int SixteenBit = 16;
      int SixtyFourBit = 64;
      int AndTest;
      boolean BoolBit = false;

      AndTest = SixteenBit & SixtyFourBit;
      SixteenBit = SixteenBit >> 3; //SixteenBit shift right for 3bit
      SixtyFourBit = SixteenBit << 3; //SixteenBit shifts lest for 3
bits
      while(Global_int == 5 && Global_Dbl == 3.21 || BoolBit == true)
      {
            SixteenBit ^= SixtyBit;
            Global_int = 1;
      }
      return AndTest;
}

int Method_Func(String m_String, Line m_Line)
{
      int if_int;
      if_int = 3;
      int pixel = 1;

      Pixel local_Pixel[30];

      boolean if_bool = false;
      boolean if_bool2 = true;

      if(if_int++ == 4)
      {
            if(if_bool ^ if_bool2 & (if_bool | (if_bool2 << if_bool))
!= if_bool)
            {
                  for(Global_dbl = 0 ; Global_dbl <= (if_int++ +
Global_int--) - 3; Global_dbl *= 1.2)
                  {
                        Bitop_Func();
                  }
            }
      }

      if(-30 == 1 || -23.3 == -24.3e+12 && if_bool == false)
      {
            Image local_image = Open("c:\image\test.jpg");
            Image local_result = new Image (local_image.getWidth(),
local_image.getHeight());
            for(int test = 0 ; (12+2*3)*4/45+2 <
local_image.getHeight(); i += pixel)
            {
                  continue;
            }
     }
}

void main(String[] args)
{
     String[] l_String = new String[10];

     Bitop_Func();
     Method_Func(true);
     Method_Func(true ^ true & (false | true) && true ^ false);
     Num_Func();
     Recall_Num_Func(-123);
     Recall_Num_Func(12+34-2947*2/4);
     Method_Func(Global_boolTest ^ Global_boolTest & (Global_boolTest |
(Global_boolTest << Global_boolTest)) != Global_boolTest);
}
```

6.3.2 Connectivity between Parser and Semantic Analyzer

6.3.2.1 Code that was not be compiled but should

The following code should be compiled since the testing code does not have any bug itself. However, the '#define' was not working. It is now fixed.

```
#define def_test 17
#define def_compare 18

void main(String args[])
{
int a = 3;
int b = 4;
int x = defineTest(a,b);
println(""+x);
println(""+def_test + " " + def_compare); // printf
17 18
}

int defineTest(int a,int b)
{
return a;
}
```

6.3.2.2 Code that compiled but should not

The following code has two mains. This would not be an appropriate way of programming. However, it compiled. We had many problems one which should not be compiled but compiled. Such as having library function as variables. Those are now fixed.

```
void main(String args[])
{
      print("This is Two mains");
}
```

```
int main(int args)
{
      print("This is Second main");
}
```

6.3.2.3 Code that should not be compiled and made an error

The following code generated NullPointerException which it should just produce an error. It is now fixed.

```
int WrongCall(){
println("Wrong int call");
return 0;
}

double WrongCall(){
println("Wrong Double call");
return 0.0;
}

String WrongCall(){
println("Wrong String call");
return "a";
}

Line Wrongcall(){
Line line = new Line(10,10,34,64);
return line;
}

void main(){
Wrongcall();
```

7 Language Restrictions

Even though our language is very robust and is capable of detecting all of the errors Java compiler would detect, there are few restrictions that the programmers have to keep in mind in order to create a run-able Photogram program. Our language is not designed to detect the violation of these restrictions, and if the programmer violates these restrictions, he/she will possibly fail to generate a run-able Photogram program. This means that even though the program will compile using our Photogram compiler, the generated Java codes will not compile using a Java compiler.

- Function declaration should be written on the same line including all of its arguments and right brace ‘{‘.
  Good Example:
  ```
  Image fft(Image src, int a, double b){
        // … function body;
  }
  ```
  Bad Example:
  ```
  Image fft(Image src,
        int a,
  ```

```
                 double b)
    {
            // … function body;
    }
```

- Any statements including function call should be written on the same line including the passing parameters for the function call.
  Good Example:
  ```
      Image src = new Image();
      src.open("test.jpg", Image.RGB_TYPE);
  ```
  Bad Example:
  ```
      Image src =
              new Image();
      src.open("test.jpg",
               Image.RGB_TYPE);
  ```

- Any statements after while(true) or if(true) statement would be treated as unreachable statement by Java compiler as an error, but our compiler won't detect these unreachable errors, so whenever you use while(true) or if(true), make sure you won't write any statements after these statements.

- Any statements inside of while(false) or if(false) statement would be treated as unreachable statement by Java compiler as an error, but our compiler won't detect these unreachable errors, so never use while(false) or if(false) statement because they are meaningless anyway.

- It is allowed to have a variable name that is same as a function name, but for some cases, it will cause problem when you use them in included files using #include macro, so we would not suggest using variable name that is same as a function name.

8 Lessons Learned

8.1 Important Learning

8.1.1 Keeping in contact with group members is very essential for successful completion of the project.
8.1.2 Don't ignore professor's advice regarding CVS.
8.1.3 Use MSN and not AOL for communicating.
8.1.4 Use JBuilder for developing your Java code.
8.1.5 Write small test programs for each stage.

8.2 Advice for future teams

8.2.1 Start early; rather start even before the semester begins.

8.2.2 If you are already looking at this report during the semester then your doomed.

8.2.3 Buy the enterprise version of JBuilder for developing your Java code, it costs only $3000.

9 Library Functions

9.1 Math Functions

9.1.1 double random()
The function returns a random double number in the range of 0 to 1

9.1.2 double cos(double radian)
The function returns cosine of the angle passed in radian units.

9.1.3 double sin(double radian)
The function returns sine of the angle passed in radian units.

9.1.4 double tan(double radian)
Returns tangent of the angle passed in radian units.

9.1.5 double acos(double val)
The function returns the angle whose cosine is val, in the range [0, pi] radians.

9.1.6 double asin(double val)
The function returns the angle whose sine is val, in the range [-pi/2, +pi/2] radians.

9.1.7 double ceil(double val)
The function returns the smallest integer value not less than val.

9.1.8 double floor(double val)
The function returns the largest integer value not greater than val.

9.1.9 double sqrt(double val)
The function returns the square root of val.

9.1.10 double abs(double val)
The function returns the absolute value of val, |val|

9.1.11 double pow(double val, double pow)
The function returns ‘val’ raised to the power ‘pow’.

9.1.12 int pow(int val, int pow)
The function returns ‘val’ raised to the power ‘`pow`’.

9.1.13 double log(double val)
The function returns the natural logarithm of val.

9.1.14 double exp(double val)
The function returns the exponential of val.

9.1.15 double pi()
The function returns the value of PI.

9.1.16 double e()
The function returns the value of natural logarithm.

9.2 Built-in Image Processing Functionalities

9.2.1 Image grayscale(Image pGImage)
Convert the source image to a gray scaled image.

9.2.2 Image quantUniform(Image pGImage)
Quantize the source image into 256 color image using an uniformly distributed color map.

9.2.3 Image quantPopulosity(Image pGImage)
Quantize the source image into 256 color image using the 256 most popular color used in this image.

9.2.4 Image ditherThreshold(Image pGImage)
Dither the source image using threshold of 0.5.

9.2.5 Image ditherRandom(Image pGImage)
Dither the source image randomly.

9.2.6 Image ditherFS(Image pGImage)
Dither the source image using Floyd-Steinberg technique.

9.2.7 Image ditherBright(Image pGImage)
Dither the source image while conserving the brightness value of the source image

9.2.8 Image ditherBlock(Image pGImage, double[][] doubleArray)
Dither the source image using a 4x4 block mask matrix.

9.2.9 Image ditherOrdered(Image pGImage)

Dither the source image using a 4x4 ordered mask matrix.

9.2.10 Image ditherCluster(Image pGImage)
Dither the source image using a 4x4 clustered mask matrix.

9.2.11 Image filterBox(Image pGImage)
Blur the source image using a 5x5 box filter.

9.2.12 Image filterBartlett(Image pGImage)
Blur the source image using a 5x5 Bartlett filter.

9.2.13 Image filterGaussian(Image pGImage)
Blur the source image using a 5x5 Gaussian filter.

9.2.14 Image filterGaussianN(Image pGImage, int size)
Blur the source image using a NxN Gaussian filter.

9.2.15 Image filterEdgeDetect(Image pGImage)
Detect the edge of the source image.

9.2.16 Image filterEnhance(Image pGImage)
Enhance the edge of the source image.

9.2.17 Image sizeHalf(Image pGImage)
Half the size of the source image

9.2.18 Image sizeDouble(Image pGImage)
Double the size of the source image

9.3 Useful Functions

9.3.1 Image[] openDir(String string)
Open all of the images in a directory.

9.3.2 int doubleToInteger(double val)
Convert double “val” to an integer

9.3.3 int maskBits(int bits, int number_bits)
Mask integer “bits” for certain number of bits

9.3.4 Image drawShape(Image pGImage, Line pGLine, Color pGColor)
Draws a line, of the color specified as input.

9.3.5 Image drawShape(Image pGImage, Oval pGOval, Color pGColor, boolean fill)
Draws an oval. “fill” will tell whether to fill the inside of the rectangle.

9.3.6 Image drawShape(Image pGImage, Rect pGRect, Color pGColor, boolean fill)
Draws a rectangle. “fill” will tell whether to fill the inside of the rectangle

9.3.7 Image drawText(Image pGImage, String string, int x, int y, Font pGFont, Color pGColor)
Draw a text onto the source image starting from coordinate (x, y) of the image with a specific font and color

9.3.8 Image resize(Image pGImage, int width, int height)
Resize the source image to a specific width and height (aspect ratio is not preserved)

9.3.9 Image paste(Image dest, Image src, int x, int y)
Paste src image to dest image starting point (x, y).

9.3.10 Image paste(Image dest, Image src, Rect rect)
Paste src image to dest image starting point (rect.x, rect.y) for rect.width wide, and rect.height height.

9.3.11 Image rotate(Image pGImage, double degree)
Rotate the source image clockwise for a specified degree

9.3.12 Image map(Image pGImage, int[] intArray)
Modify the color of the image using the given colorMap.

9.3.13 Image convol(Image pGImage, double[][] doubleArray)
Convolute the given image with a given mask.

9.3.14 void print(String string)
Prints the string specified by the user.

9.3.15 void println(String string)
Prints the string specified by the user.

10 References

- http://www.photomosaic.com/rt/fineart.htm
- 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.
- Ritchie, Dennis M. *C Reference Manual*. Bell Telephone Laboratories, 1975.

11 Appendix

11.1 Photogrammar.g

```
// Author: Richard, Neesha
class PGAntlrLexer extends Lexer;

options
      {
      k = 2;
      charVocabulary = '\3'..'\377';
      testLiterals = false;
      exportVocab = PGAntlr;
      }

protected
LETTER   : 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT   :   '0'..'9';

WS      : (' ' | '\t')+          { $setType(Token.SKIP); }
        ;

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); }
        ;

COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                 )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )                      { $setType(Token.SKIP); }
        ;

LPAREN  : '(';
RPAREN  : ')';
MULT    : '*';
PLUS    : '+';
MINUS   : '-';
DIV     : '/';
MOD     : '%';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
LBRK    : '[';
RBRK    : ']';
ASGN    : '=';
COMMA   : ',';
DOT     : '.';
PLUSEQ  : "+=";
```

```
MINUSEQ : "-=";
MULTEQ  : "*=";
DIVEQ   : "/=";
MODEQ   : "%=";
GTE     : ">=";
LTE     : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
NOT     : '!';
OR      : "||";
AND     : "&&";
BOR     : '|';
BAND    : '&';
BXOR    : '^';
OREQ    : "|=";
ANDEQ   : "&=";
XOREQ   : "^=";
MACRO   : '#';
LSHIFT  : "<<";
RSHIFT  : ">>";
PPLUS   : "++";
MMINUS  : "--";
COLON : ':';

ID  options { testLiterals = true; }
        : ('a'..'z' | 'A'..'Z')  (LETTER|DIGIT)*
        ;

NUMBER  : (DIGIT)+ ('.' (DIGIT)*)? (('E'|'e') ('+'|'-')? (DIGIT)+)?
      ;

STRING  : '"'!
            (  ~('"' | '\n' )
             | ('"'!'"')
            )*
          '"'!
        ;

class PGAntlrParser extends Parser;

options{
    k = 3;
    buildAST = true;
    exportVocab = PGAntlr;
    ASTLabelType = "CommonASTWithLines";
}

tokens {
   PROGRAM;
   STATEMENT;
   FUNCTION;
   DECLARATION;
   ARRAY;
   FUNC_VAR_LIST;
   VAR_LIST;
   EXPR_LIST;
   CASE_LIST;
   CASE;
   FUNC_CALL;
   FOR_CON;
   ASSIGNMENT;
   EXPR1;
   EMPTY_ARRAY;
   GLOBAL;
   SWITCH;
   EMPTY_DARRAY;
   D_ARRAY;
   D_ARRAY_R;
   UMINUS;
   OBJ_CALL;
}

program
        : ( function | macro | global )* EOF!
            { #program = #([PROGRAM, "PROGRAM"], program); }
        ;

function
      :( ( ((data_type|"void") ID) | ((data_type|"void")empty_darray))
LPAREN! (var_list)? RPAREN! statement )
            { #function = #([FUNCTION,"FUNCTION"], function); }
      ;

macro
      : include_macro
      | define_macro
      ;

global
      : global_stmt
        ;

global_stmt
      : data_type asgn_assign (COMMA! asgn_assign)* SEMI!
      { #global_stmt = #([GLOBAL, "GLOBAL"], global_stmt); }
      ;

primitive
        : "int"
      | "double"
        | "boolean"
        ;

builtin_obj
      : "Line"
        | "Color"
        | "Pixel"
        | "Rect"
        | "Oval"
        | "String"
        | "Image"
        | "Font"
      ;

data_type
      : (primitive | builtin_obj)
        ;

statement
        : for_stmt
        | if_stmt
        | break_stmt
        | continue_stmt
        | return_stmt
        | assignment_stmt
        | function_stmt
        | while_stmt
        | switch_stmt
        | LBRACE! (statement)* RBRACE!
            {#statement = #([STATEMENT,"STATEMENT"], statement); }
        ;

assignment_stmt
      : assignment SEMI!
        ;

assignment
      : data_type asgn_assign ( COMMA! asgn_assign )*
           {#assignment = #([DECLARATION,"DECLARATION"], assignment); }
      | ( ID | array ) ( ( ( (ASGN^ | ANDEQ^ | XOREQ^ |  OREQ^ |
PLUSEQ^ | MINUSEQ ^ | MULTEQ^ | DIVEQ^ | MODEQ^) expression )
            | ( MMINUS^ | PPLUS^ ) )
            | ((object_call){#assignment = #([OBJ_CALL,
"OBJ_CALL"],assignment);} ( (ASGN^ | ANDEQ^ | XOREQ^ |  OREQ^ | PLUSEQ^
| MINUSEQ ^ | MULTEQ^ | DIVEQ^ | MODEQ^) expression)? )
              )
      {#assignment = #([ASSIGNMENT,"ASSIGNMENT"], assignment); }
      ;

asgn_assign
      : ID (ASGN^ expression )?
      | array ASGN^ expression
      ;

function_stmt
      : func_call SEMI!
      ;

switch_stmt
      : "switch"^ LPAREN! expression RPAREN! LBRACE!
         case_list
        (default_stmt)? RBRACE!
      ;

case_list
      : (case_stmt)*
       {#case_list = #([CASE_LIST, "CASE_LIST"], case_list);}
      ;
```

```
case_stmt
      : "case"! (NUMBER | STRING) COLON! (statement)*
            {#case_stmt = #([CASE,"CASE"], case_stmt);}
      ;

default_stmt
      : "default"^ COLON! (statement)*
      ;

while_stmt
        : "while"^ LPAREN! expression RPAREN! statement
        ;

for_stmt
        : "for"^ LPAREN! for_con RPAREN! statement
        ;

for_con
        : (assignment)? SEMI! (expression)? SEMI! (assignment)?
            {#for_con = #([FOR_CON,"FOR_CON"], for_con); }
        ;

if_stmt
        : "if"^ LPAREN! expression RPAREN! statement
            (options {greedy = true;}: "else"! statement )?
        ;

break_stmt
        : "break"^ SEMI!
        ;

continue_stmt
        : "continue"^ SEMI!
        ;

return_stmt
        : "return"^ (expression)? SEMI!
        ;

include_macro
        : MACRO! "include"^ STRING
        ;

define_macro
        : MACRO! "define"^ ID ( NUMBER | STRING )
        ;

func_call
        : (ID LPAREN! (expr_list)? RPAREN!)
          {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
        ;

expr_list
        :(expr1) (COMMA! (expr1))*
      {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
      ;
```

```
expr1 :
        expression ( assignment_op expression )?
        {#expr1 = #([EXPR1,"EXPR1"], expr1); }
      ;

var_list
        : data_type (ID|empty_array) ( COMMA! data_type 
(ID|empty_array) )*
        { #var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

assignment_op
      : ASGN^ | ANDEQ^ | XOREQ^ |  OREQ^ | PLUSEQ^ | MINUSEQ ^ | 
MULTEQ^ | DIVEQ^ | MODEQ^
      ;

expr_term
      : l_value (assignment_op expression)?
      ;

expression
      : assignment_term
      ;

assignment_term
        : logic_term ( OR^ logic_term )*
        ;

logic_term
        : bor_expr ( AND^ bor_expr )*
        ;

bor_expr
      : bxor_expr( (BOR^) bxor_expr )*
      ;

bxor_expr
      : band_expr( (BXOR^) band_expr )*
      ;

band_expr
      : eq_factor ( (BAND^) eq_factor )*
      ;

eq_factor
        : relat_expr ( (EQ^ | NEQ^) relat_expr )?
        ;

relat_expr
        : shift_expr ( (GTE^ | LTE^ | GT^ | LT^) shift_expr )?
        ;

shift_expr
        : arith_expr ( (LSHIFT^ | RSHIFT^) arith_expr )*
        ;

arith_expr
```

```
        : arith_term ( (PLUS^ | MINUS^) arith_term )*
        ;

arith_term
        : arith_factor ( (MULT^ | DIV^ | MOD^) arith_factor )*
        ;

arith_factor
        : (NOT^)* r_value (MMINUS^ | PPLUS^)?
        ;

r_value
        : l_value
        | func_call
        | NUMBER
        | STRING
        | data_type DOT! ID
            {#r_value = #([OBJ_CALL, "OBJ_CALL"],r_value);}
        | "true"^
        | "false"^
        | "null"^
        | LPAREN! expression RPAREN!
        | MINUS! r_value
            {#r_value = #([UMINUS,"UMINUS"], r_value); }
        | "new"^ (builtin_obj(LPAREN! (expr_list)? RPAREN!) | data_type
d_array)
        ;

l_value
        : ( ID | array ) (object_call {#l_value = #([OBJ_CALL,
"OBJ_CALL"],l_value);})?
        ;

object_call
      : ( DOT! (func_call | ID) )
      ;

array
      : ID (LBRK (expression)? RBRK! )+
        {#array = #([ARRAY,"ARRAY"], array); }
      ;

d_array
      : (LBRK (expression)? RBRK! )+
        {#d_array = #([D_ARRAY_R,"D_ARRAY_R"], d_array); }
      ;

empty_array
      : ID (LBRK RBRK! )+
        {#empty_array = #([ARRAY,"ARRAY"], empty_array); }
      ;

empty_darray
      : (LBRK RBRK! )+ ID
        {#empty_darray = #([D_ARRAY,"D_ARRAY"], empty_darray); }
      ;
```

11.2 Photowalker.g

```
// Author: Richard
{
import java.io.*;
import java.util.*;
}

class PGAntlrWalker extends TreeParser;
options
      {
      importVocab = PGAntlr;
      ASTLabelType = "CommonASTWithLines";
      }

{
SemanticAnalyzer sa; //contains all functions and methods to determine
semantic errors
int current_line=0;
String errMsg = ""; // contains error messages from the semantic
analyzer
}

expr returns [ PGDataType r ] //all matches return either nothing or a
PGDataType
      {
      PGDataType a=null, b=null; //used temporarily in matches to
determine semantic errors
      Object[] c=null; //used to hold either a list of variables or
expressions
      r=null; //the resulting datatype when evaluated by the semantic
analyzer
      }

      //unary operators
        : #(NOT a=expr)                {
                           try
                               {r = sa.checkType( a,
SemanticAnalyzer.NOT );}
                                   catch(PGException pe)
                                       {errMsg+="linulte
"+current_line+": "+pe.getMessage()+"\n";}
                                   }
        | #(PPLUS a=expr)              {
                           try
                               { r = sa.checkType( a,
SemanticAnalyzer.PPLUS ); }
                           catch(PGException pe)
                               {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(MMINUS a=expr)             {
                           try
                              { r = sa.checkType( a,
SemanticAnalyzer.MMINUS ); }
                           catch(PGException pe)
```

```
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(UMINUS a=expr)       {
                           try
                              { r = sa.checkType( a,
SemanticAnalyzer.MINUS ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }

        //binary operators
        | #(GTE a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.GTE ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(LTE a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.LTE ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(GT a=expr b=expr)      {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.GT ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(LT a=expr b=expr)      {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.LT ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(EQ a=expr b=expr)      {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.EQ ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(NEQ a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.NEQ ); }
```

```
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(PLUS a=expr b=expr)    {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.PLUS ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(MINUS a=expr b=expr)   {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.MINUS ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(MULT a=expr b=expr)    {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.MULT ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(DIV a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.DIV ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(MOD a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.MOD ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(OR a=expr b=expr)      {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.OR ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(AND a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.AND ); }
                           catch(PGException pe)
```

```
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(BOR a=expr b=expr)     {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.BOR ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(BAND a=expr b=expr)    {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.BAND ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(BXOR a=expr b=expr)    {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.BXOR ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(LSHIFT a=expr b=expr)  {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.LSHIFT ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }
        | #(RSHIFT a=expr b=expr)  {
                           try
                              { r = sa.checkType( a, b,
SemanticAnalyzer.RSHIFT ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }

      //assignment operators
        | #(ASGN a=expr b=expr)    {
                            try
                              {r = sa.assign(a, b, false,
SemanticAnalyzer.ASGN, current_line );}
                            catch (PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}

                            }
        | #(PLUSEQ a=expr b=expr)  {
                           try
```

```
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.PLUSEQ, current_line ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(MINUSEQ a=expr b=expr) {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.MINUSEQ, current_line ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(MULTEQ a=expr b=expr)  {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.MULTEQ, current_line ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(DIVEQ a=expr b=expr)   {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.DIVEQ, current_line ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(MODEQ a=expr b=expr)   {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.MODEQ, current_line ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(OREQ a=expr b=expr)    {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.OREQ, current_line ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(ANDEQ a=expr b=expr)   {
                           try
                              { r = sa.assign(a, b, false, 
SemanticAnalyzer.ANDEQ, current_line ); }
                           catch(PGException pe)
                                {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
                                   }
        | #(XOREQ a=expr b=expr)   {
                           try
```

```
                              { r = sa.assign(a, b, false,
SemanticAnalyzer.XOREQ, current_line ); }
                           catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                                   }

                                   //TODO:NEESHA

        //statements
        | #(FUNC_CALL func_call_id:ID (c=expr_list)?)
            {
            current_line = func_call_id.getLine();
            {
            try
                  { r = (PGDataType)(sa.getFunc(func_call_id.getText(),
c, false)); }
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }
        | #(OBJ_CALL ((o_id1:data_type o_id2:ID)
            {
            a = new PGDataType(o_id2.getText());
            a.setType(o_id1.getText());
            {
            try
                  {r = sa.checkConst(a,current_line);}
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }
            |((o_id3:ID {a = new PGDataType(o_id3.getText());}
              |
              #(ARRAY {int i=0;} //initialize variable to count number
of dimensions in array
                  array_id3:ID {current_line = array_id3.getLine();}
                  (LBRK {i++;} //every left bracket it sees increases
the number of dimensions of array by 1
                        (b=expr
                        {
                        try
                              {sa.checkArrayIndex(b);}
                        catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }

                        )?
                  )+
                   {a = new PGArray(i, array_id3.getText() );}
           )
              )
                  (#(FUNC_CALL object_func:ID (c=expr_list)?)
                  {
```

```
                  current_line = object_func.getLine();
                  try
                        {r = sa.checkObjectCall(a, new
PGDataType(object_func.getText()), c, current_line);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                  | (o_id4:ID)
                    {
                    b = new PGDataType(o_id4.getText());
                    current_line = o_id4.getLine();
                    try
                        {
                        r = sa.checkConst(a, b, current_line);
                        }
                    catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                    }
                  )
                 )
             )
           )
        | #("return" (a=expr)?)
        {
        try
            {
            if (a==null)
                  {
                  a = new PGDataType();
                  a.setType("void");
                  }
            sa.checkReturn(a);
            }
            catch(PGException pe)
            {errMsg+="line "+current_line+": "+pe.getMessage()+"\n";}
        }
        | #(DECLARATION  dec_type:data_type //special cases required
below as one can also assign in declaration statement
            (#(ASGN a=expr b=expr) //place variable into symbol table
first before evaluating assign
                  {
                        if (a instanceof PGArray)

      ((PGArray)a).setArrayType(dec_type.getText());
                        else
                              a.setType(dec_type.getText());
                        {
                        try
                        {sa.putVar(a, false, false, current_line);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }

                  {
```

```
                  try
                        {sa.assign(a,b, true, SemanticAnalyzer.ASGN,
current_line);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }

                  }
             | dec_id:ID //proceed normally if ID is found instead of
assignment
                        {
                        current_line = dec_id.getLine();
                        a=new PGDataType(dec_id.getText());
                        a.setType(dec_type.getText());

                        {
                        try
                              {sa.putVar(a, false, false,
current_line);}
                        catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                        }
             | #(ARRAY {int i=0;} array_id1:ID (LBRK {i++;} )+)
//proceed normally if Array is found instead of assignment
                        {
                        current_line = array_id1.getLine();
                        a = new PGArray(i, array_id1.getText());
                        ((PGArray)a).setArrayType(dec_type.getText());
                        {
                        try
                              {sa.putVar(a, false, false,
current_line);}
                        catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                        }
             )+
            )
        | #(ASSIGNMENT a=expr) //assignment can only be done once per
statement
        | #("new" n_type:data_type ((c=expr_list)?
            {
            current_line = n_type.getLine();
            {
            try
                  {r = sa.getFunc(n_type.getText(), c, false);}
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }
            | (#(D_ARRAY_R {int k=0;}
                  (LBRK {k++;}
                  a=expr
                  {
                  try
                        {sa.checkArrayIndex(a);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                  )+
               )
                  {
                  r = new PGArray(k, "unknown");
                  ((PGArray)r).setArrayType(n_type.getText());
                  }
                  ))
          )
        | #("switch" {sa.enterScope(null, false, true);} //when
"switch" node is matched, enter a new scope
            a=expr {
                  try
                        {sa.checkSwitch(a,current_line);}
                  catch (PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
            #(CASE_LIST (case_)*
             )
            (#("default" {
                  try
                        {sa.checkDefault();}
                  catch (PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }(def_stmt:. {expr(#def_stmt);})*))? //execute all
statements in default
           )
            {sa.leaveScope();sa.endSwitch();} //when switch is done,
leave scope and clear cases defined
        | break_stmt:"break"
            {
            try
                  {
                  current_line = break_stmt.getLine();
                  sa.checkBreak();
                  }
            catch (PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
            }
        | continue_stmt:"continue"
            {
            try
                  {
                  current_line = continue_stmt.getLine();
                  sa.checkContinue();
                  }
            catch (PGException pe)
```

```
                  {errMsg+="line "+current_line+": 
"+pe.getMessage()+"\n";}
            }

        | #("while" {sa.enterScope(null, true, false);} a=expr (b = 
expr | #(STATEMENT (stmt_block2:. {expr(#stmt_block2);})*) ))
        {
        try
                  {sa.checkWhile(a);}
        catch(PGException pe)
            {errMsg+="line "+current_line+": "+pe.getMessage()+"\n";}
      sa.leaveScope();
        }
        | #("if" a=expr ({sa.enterScope(null, false, false);} b = expr 
{sa.leaveScope();}  | statement) ({sa.enterScope(null, false, false);} 
b = expr {sa.leaveScope();} | statement)?)
        {
        try
                  {sa.checkIf(a);}
        catch(PGException pe)
            {errMsg+="line "+current_line+": "+pe.getMessage()+"\n";}
        }
        | #("for" {sa.enterScope(null, true, false);} //when "for" node 
is matched, enter a new scope
            a = expr  (b= expr | #(STATEMENT (stmt_block1:. 
{expr(#stmt_block1);})*)) //execute all statements in for loop
           )
            {sa.leaveScope();} //when for is done, leave scope
        | #(FOR_CON a = expr b = expr)    //second condition of for 
loop must be boolean
        {
        try
            {sa.checkFor(b);}
      catch(PGException pe)
            {errMsg+="line "+current_line+": "+pe.getMessage()+"\n";}
        }
        //"base" cases
        | d_type:data_type (a=expr) //used in functions where a 
datatype and ID follow after function node
            {
            if (a instanceof PGArray) //PGArray
                  ((PGArray)a).setArrayType(d_type.getText());
            else //or PGDataType
                  a.setType(d_type.getText());
            r=a;
            }
        | #(ARRAY {int i=0;} //initialize variable to count number of 
dimensions in array
            array_id:ID {current_line = array_id.getLine();}
                  (LBRK {i++;} //every left bracket it sees increases 
the number of dimensions of array by 1
                        (a=expr
                        {
                        try
                              {sa.checkArrayIndex(a);}
                        catch(PGException pe)
```

```
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }

                        )?
                  )+
            )
             {
             r = new PGArray(i, array_id.getText());
             }
        | #(D_ARRAY {int j=0;} //return array for functions, declared
differently then arrays of variables
             (LBRK {j++;})+ array_id2:ID {current_line =
array_id2.getLine();}
            )
             {
             r = new PGArray(j, array_id2.getText());
             }
      | id:ID //create PGDataType for ID
             {
             current_line = id.getLine();
             r = new PGDataType(id.getText());
             }
        | num:NUMBER //creates appropriate PGDataType for Number
             {
             current_line = num.getLine();
             if ( (num.getText().indexOf(".")>-1) //search through chars
to determine if double
                   || (num.getText().indexOf("e")>-1)
                   || (num.getText().indexOf("E")>-1)
                )
                   {
                   r = new PGDataType();
                   r.setType("double");
                   }
             else //otherwise it is an integer
                   {
                   r = new PGDataType();
                   r.setType("int");
                   }
             }
             | str:STRING //creates appropriate PGDataType for String
                   {
                   current_line = str.getLine();
                   r = new PGDataType();
                   r.setType("String");
                   }
             | ("true"|"false") //creates appropriate PGDataType for
booleans
                   {
                   r = new PGDataType();
                   r.setType("boolean");
                   }

        | #(PROGRAM (stmt:. {expr(#stmt);} //PROGRAM is first node
encountered, go through all its children
                 )*
                )
        | #(GLOBAL stmt1:.) //ignore globals, taken care of in
prewalker
        | #("define" define_id:ID
(define_num:NUMBER|define_string:STRING)) //ignore defines
        | #("include" file:STRING)
        | #(FUNCTION a=expr (c=var_list)?
            {
            sa.enterFunction();
            PGFunction func = null;
            try{
                  func = (PGFunction)sa.getFunc(a.getID(),c,true);
            }
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
            sa.enterScope(func, false, false);//entering scope of a
particular function
            if (c!=null)
                  {
                  for (int x=0; x<c.length; x++) //put functions
arguments into current scope's symbol table
                  {
                  try
                        {sa.putVar((PGDataType)c[x], false, true,
current_line);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                  }
            }
            #(STATEMENT (stmt_block:. {expr(#stmt_block);})* //go
through and check all of function's children for semantic errors
           )
              { try
                  {
                  sa.leaveFunction();
                  }
                catch (PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  sa.leaveScope();} //leave scope after function is
done
           )

        ;

case_  //used to denote a single case of switch structure
      {
      PGDataType a; //used to get id of case
      }
      : #(CASE (a_num:NUMBER|a_str:STRING )
                  {
                  if (a_num!=null)
                        {
                        current_line = a_num.getLine();
                        if ( (a_num.getText().indexOf(".")>-1) //search
through chars to determine if double
                             || (a_num.getText().indexOf("e")>-1)
                             || (a_num.getText().indexOf("E")>-1)
                           )
                              {
                              a = new PGDataType(a_num.getText());
                              a.setType("double");
                              }
                        else //otherwise it is an integer
                              {
                              a = new PGDataType(a_num.getText());
                              a.setType("int");
                              }
                        }
                  else
                        {
                        current_line=a_str.getLine();
                        a = new PGDataType(a_str.getText());
                        a.setType("String");
                        }
                  try
                        {sa.putCase(a, current_line);} //add each case
to a list
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
             (case_stmt:. {current_line =
case_stmt.getLine();expr(#case_stmt);} //evaluate all statements in
case
             )*
          )
      ;

var_list returns [Object[] r] //evaluates a list of variables and
returns an array of PGDataType
      {
      Vector v = new Vector();;
      PGDataType a;
      r=null;
      }

      :
      #(VAR_LIST        {v = new Vector();}
            (a=expr         {v.add(a);}
            )*
       )                {
                        r = v.toArray();
                        }
      ;

expr_list returns [Object[] r] //evaluates a list of expressions
returns an array of PGDataType
      {
      Vector v;
      PGDataType a;
      r=null;
      }

      : #(EXPR_LIST           {v = new Vector();}
            (#(EXPR1 a=expr){v.add(a);}
            )*
         )              {r = v.toArray();}
      ;

data_type //various data types available
      : "int"
      | "double"
      | "boolean"
      | "void"
      | "Line"
      | "Color"
      | "Pixel"
      | "Rect"
      | "Oval"
      | "String"
      | "Image"
      | "Font"
      ;

statement
      :
      #(STATEMENT {sa.enterScope(null, false, false);}  //enter scope
when entering a brace of statements
            (stmt_block:. {expr(#stmt_block);} //evaluate all
statements within braces
              )*
       )
            {sa.leaveScope();} //leave scope when all statements have
been evaluated
      ;
```

11.3 PrePhotowalker.g

```
// Author: Richard
{
import java.io.*;
import java.util.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
}

class PrePGAntlrWalker extends TreeParser;
options
      {
      importVocab = PGAntlr;
      ASTLabelType = "CommonASTWithLines";
      }

{
```

```
public SemanticAnalyzer sa; //contains all functions and methods to
determine semantic errors
int current_line=0;
String errMsg = ""; // contains error messages from the semantic
analyzer
File mainFile;
File currFile;
}

expr returns [ PGDataType r ] //all matches return either nothing or a
PGDataType
      {
      PGDataType a=null, b=null; //used temporarily in matches to
determine semantic errors
      Object[] c=null; //used to hold either a list of variables or
expressions
      r=null; //the resulting datatype when evaluated by the semantic
analyzer
      }
      //assignment
        : #(ASGN a=expr b=expr)
      { sa.assign(a,b,false,SemanticAnalyzer.ASGN, current_line); }

      //statements
        | #(GLOBAL glob_type:data_type //special cases required below
as one can also assign in global declaration statement
            (#(ASGN a=expr b=expr) //place variable into symbol table
first before evaluating assign
                        {
                        if (a instanceof PGArray)
                  ((PGArray)a).setArrayType(glob_type.getText());
                        else
                              a.setType(glob_type.getText());
                        {
                        try
                             {sa.putVar(a, true, false, current_line);}
                        catch(PGException pe)
                          {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }

                        {
                        try
                              {sa.assign(a,b,true,SemanticAnalyzer.ASGN,
current_line);}
                        catch(PGException pe)
                          {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }
                        }
               | global_id:ID //proceed normally if ID is found instead
of assignment
                        {
                        a = new PGDataType(global_id.getText());
                        a.setType(glob_type.getText());
                        current_line=global_id.getLine();
                        {
```

```
                        try
                             {sa.putVar(a, true, false, current_line);}
                        catch(PGException pe)
                          {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }
                        }
                  | #(ARRAY {int i=0;} array_id1:ID (LBRK {i++;} )+)
//proceed normally if Array is found instead of assignment
                        {
                        a = new PGArray(i, array_id1.getText());
                        ((PGArray)a).setArrayType(glob_type.getText());
                        current_line=array_id1.getLine();
                        {
                        try
                             {sa.putVar(a, true, false, current_line);}
                        catch(PGException pe)
                          {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }
                        }
                  )+
              )
           | #("define" define_id:ID
(define_num:NUMBER|define_string:STRING))
            {
            current_line = define_id.getLine();
            String value = ""; //used to hold the value of define
statement
            if (define_num != null) //check to see if int or double
                  {
                  if ( (define_num.getText().indexOf(".")>-1)
                       || (define_num.getText().indexOf("e")>-1)
                       || (define_num.getText().indexOf("E")>-1)
                     )
                        {
                        a=new PGDataType(define_id.getText());
                        a.setType("double");
                        current_line = define_num.getLine();
                        }
                  else
                        {
                        a=new PGDataType(define_id.getText());
                        a.setType("int");
                        current_line = define_num.getLine();
                        }
                  value = define_num.getText();
                  }
            else  //otherwise it is a string
                  {
                  a=new PGDataType(define_id.getText());
                  a.setType("String");
                  value = define_string.getText();
                  current_line = define_string.getLine();
                  }
            {
            try
```

```
                  {sa.putDefine(a, value, currFile);} //place into
symbol table
            catch(PGException pe)
                    {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }
        | #("include" file:STRING)
            {
            try
                  {
                  File includeFile = new File(file.getText());
                  BufferedReader pgFile = new BufferedReader(new
FileReader(includeFile));

                  PGAntlrLexer lexer = new PGAntlrLexer(pgFile);
                  PGAntlrParser parser = new PGAntlrParser(lexer);

      parser.setASTNodeClass(CommonASTWithLines.class.getName());
                  parser.program();

                  CommonASTWithLines parseTree = (CommonASTWithLines)
parser.getAST();

                  PrePGAntlrWalker prewalker = null;
                  PGAntlrWalker walker = null;

                  SemanticAnalyzer sa1 = new SemanticAnalyzer();

                  prewalker = new PrePGAntlrWalker();
                  prewalker.mainFile = mainFile;
                  prewalker.currFile = includeFile;
                  prewalker.sa = sa;

      prewalker.setASTNodeClass(CommonASTWithLines.class.getName());
                  prewalker.expr(parseTree);
                  System.out.println(prewalker.errMsg);

                  prewalker.sa = sa1;
                  prewalker.expr(parseTree);

                  walker = new PGAntlrWalker();
                  walker.sa = sa1;

      walker.setASTNodeClass(CommonASTWithLines.class.getName());
                  walker.expr(parseTree);
                        System.out.println(walker.errMsg);

                        if(prewalker.errMsg.equals("") &&
walker.errMsg.equals(""))
                              {
                        PGCodeGenerator codeGenerator = new
PGCodeGenerator(mainFile, includeFile, sa.getEnv());
                        codeGenerator.generateJavaCode();
                        sa.refreshEnv();
                        sa1.refreshEnv();
                              }
                    }
            catch (Exception e)
                  {
                  System.out.println("File doesn't exist");
                  }
            }

        | #("new" n_type:data_type ((c=expr_list)?
            {
            current_line = n_type.getLine();
            {
            try
                  {r = sa.getFunc(n_type.getText(), c, false);}
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }
            | (#(D_ARRAY_R {int k=0;}
                  (LBRK {k++;}
                  a=expr
                  {
                  try
                        {sa.checkArrayIndex(a);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                  )+
               )
                  {
                  r = new PGArray(k, "unknown");
                  ((PGArray)r).setArrayType(n_type.getText());
                  }
                  ))
          )


        //function declaration
        | #(FUNCTION a=expr (c=var_list)? #(STATEMENT (stmt_block:.)*))
            {
            b=new PGFunction(a, a.getID(), c);
            ((PGFunction)b).setClassName(currFile);
            {
            try
                  {sa.putFunc(b, current_line);}
            catch(PGException pe)
                    {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                }
            }

        | #(PROGRAM (stmt:. {expr(#stmt);} //iterate through all of
PROGRAM nodes children
                    )*
           )
            //{sa.printEnv();}
        | #(OBJ_CALL ((o_id1:data_type o_id2:ID)
            {
            a = new PGDataType(o_id2.getText());
            a.setType(o_id1.getText());
            {
            try
                  {r = sa.checkConst(a,current_line);}
            catch(PGException pe)
                  {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                {
            }
            |((o_id3:ID {a = new PGDataType(o_id3.getText());}
              |
              #(ARRAY {int i=0;} //initialize variable to count number
of dimensions in array
                  array_id3:ID {current_line = array_id3.getLine();}
                  (LBRK {i++;} //every left bracket it sees increases
the number of dimensions of array by 1
                        (b=expr
                        {
                        try
                              {sa.checkArrayIndex(b);}
                        catch(PGException pe)
                              {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                        }

                        )?
                  )+
                   {a = new PGArray(i, array_id3.getText() );}
           )
              )
                  (#(FUNC_CALL object_func:ID (c=expr_list)?)
                  {
                  current_line = object_func.getLine();
                  try
                        {r = sa.checkObjectCall(a, new
PGDataType(object_func.getText()), c, current_line);}
                  catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                  }
                  | (o_id4:ID)
                    {
                    b = new PGDataType(o_id4.getText());
                    current_line = o_id4.getLine();
                    try
                        {r = sa.checkObjectCall(a, b, null,
current_line);}
                    catch(PGException pe)
                        {errMsg+="line "+current_line+":
"+pe.getMessage()+"\n";}
                    }
                  )
                 )
              )
            )

      //"base" cases
      | d_type:data_type (a=expr) //used in functions where a datatype
and ID follow after function node
            {
            current_line = d_type.getLine();
            if (a instanceof PGArray)
                  ((PGArray)a).setArrayType(d_type.getText());
            else
                  a.setType(d_type.getText());
            r=a;
            }
        | #(ARRAY {int i=0;} //initialize variable to count number of
dimensions in array
            array_id:ID (LBRK {i++;current_line=array_id.getLine();}
//every left bracket it sees increases the number of dimensions of
array by 1
                  )+
            )
            {
            r = new PGArray(i, array_id.getText());
            }
        | #(D_ARRAY {int j=0;} //return array for functions, declared
differently then arrays of variables
            (LBRK {j++;})+ array_id2:ID
{current_line=array_id2.getLine();}
            )
            {
            r = new PGArray(j, array_id2.getText());
            }
      | id:ID //create PGDataType for ID
            {
            current_line=id.getLine();
            r = new PGDataType(id.getText());
            }
        | num:NUMBER //creates appropriate PGDataType for Number
            {
            current_line = num.getLine();
            if ( (num.getText().indexOf(".")>-1) //search through chars
to determine if double
                 || (num.getText().indexOf("e")>-1)
                 || (num.getText().indexOf("E")>-1)
               )
                  {
                  r = new PGDataType();
                  r.setType("double");
                  }
            else //otherwise it is an integer
                  {
                  r = new PGDataType();
                  r.setType("int");
                  }
            }
            | str:STRING //creates appropriate PGDataType for String
                  {
```

```
            current_line = str.getLine();
                  r = new PGDataType();
                  r.setType("String");
                  }
            | ("true"|"false") //creates appropriate PGDataType for
booleans
                  {
                  r = new PGDataType();
                  r.setType("boolean");
                  }
        ;

var_list returns [Object[] r] //evaluates a list of variables and
returns an array of Strings
      {
      Vector v;
      PGDataType a;
      r=null;
      }

      :
      #(VAR_LIST        {v = new Vector();}
            (a=expr           {v.add(a);}
            )*
       )                {
                        r = v.toArray();
                        }
      ;

expr_list returns [Object[] r] //evaluates a list of expressions
returns an array of PGDataType
      {
      Vector v;
      PGDataType a;
      r=null;
      }

      : #(EXPR_LIST           {v = new Vector();}
            (#(EXPR1 a=expr){v.add(a);}
            )*
         )              {r = v.toArray();}
      ;


data_type //various data types available
      : "int"
      | "double"
      | "boolean"
      | "void"
      | "Line"
      | "Color"
      | "Pixel"
      | "Rect"
      | "Oval"
      | "String"
      | "Image"
      | "Font"
```

```
    ;
```

11.4 PGDataType

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */
public class PGDataType {

    private String type;
    private String id;
    private String special_field;

    public PGDataType() {
        type = "unknown";
        id = "unknown";
        special_field = "";
    }

    public PGDataType(String id) {
        this.id = id;
        type = "unknown";
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getID() {
        return id;
    }

    public void setSpecialField(String sp_field){
        this.special_field = sp_field;
    }

    public String getSpecialField(){
        return special_field;
    }

    public String toString(){
        return type;
    }
}
```

11.5 PGArray

```
package photogram;
```

```
class PGArray extends PGDataType {
    private int dimensions;
    private String array_type;

    PGArray() {
        dimensions = 0;
        array_type = "unknown";
    }

    PGArray(int dimensions, String id) {
        super(id);
        this.dimensions = dimensions;
        array_type = "unknown";
    }

    public String getType() {
        return "array";
    }

    public String getArrayType() {
        return array_type;
    }

    public void setArrayType(String type) {
        array_type = type;
    }

    public int getDimensions() {
        return dimensions;
    }

    public String toString(){
        String s = array_type;
        for(int i = 0; i < dimensions; i++)
            s += "[]";
        return s;
    }
}
```

11.6 PGFunction

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC.</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.io.*;

public class PGFunction extends PGDataType{

    private String name;
    private PGDataType return_type;
    private PGDataType [] arg_type;
    private String className;
    public PGFunction(Object return_type, String name, Object [] arg_type) {
        this.name = name;
        this.className = "";
        PGDataType temp = (PGDataType) return_type;
        if(temp instanceof PGArray){
            this.return_type   =   new   PGArray(((PGArray)temp).getDimensions(),
temp.getID());

((PGArray)this.return_type).setArrayType(((PGArray)temp).getArrayType());
        }
        else{
            this.return_type = new PGDataType(temp.getID());
            this.return_type.setType(temp.getType());
        }
        if(arg_type != null){
            this.arg_type = new PGDataType[arg_type.length];

            for (int i = 0; i < arg_type.length; i++) {
                temp = (PGDataType) arg_type[i];
                if (temp instanceof PGArray) {
                    this.arg_type[i]     =     new     PGArray(     (     (PGArray)
temp).getDimensions(), temp.getID());
                    (   (PGArray)this.arg_type[i]).setArrayType(   (   (PGArray)
temp).getArrayType());
                }
                else {
                    this.arg_type[i] = new PGDataType(temp.getID());
                    this.arg_type[i].setType(temp.getType());
                }
            }
        }
        else
            this.arg_type = null;
    }

    public String getType(){
        return "function";
    }

    public String getID(){
        return name;
    }

    public PGDataType getReturnType(){
        return return_type;
    }

    public PGDataType [] getArgTypes(){
        return arg_type;
    }

    public void setClassName(File file){
        this.className = file.getName();
    }

    public String getClassName(){
        return className;
    }

    public String toString(){
        return return_type.toString();
    }
}
```

11.7 PGImage

```
package photogram;

import java.awt.*;
import java.awt.image.*;

import java.io.*;
import javax.imageio.*;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGImage{

    public final static int JPEG = 0;
    public final static int PNG = 1;

    public final static int RGB_TYPE = BufferedImage.TYPE_INT_RGB;
    public final static int ARGB_TYPE = BufferedImage.TYPE_INT_ARGB;

    /** File name of the opened file **/
    private String filename;

    /** A buffered image associated with this image object **/
    private BufferedImage bi;

    /** Width of this image **/
    private int width;

    /** Height of this image **/
    private int height;

    /** The color type of the buffered image **/
    private int type;

    private PGImage(BufferedImage bi, int width, int height){
        this.filename = "";
        this.bi = bi;
        if(width < 0)
            this.width = 0;
        else
            this.width = width;

        if(height < 0)
            this.height = 0;
        else
            this.height = height;
    }

    public PGImage() {
        this(new BufferedImage(1, 1, RGB_TYPE), 1, 1);
    }

    public PGImage(int width, int height) {
        this(new BufferedImage(width, height, RGB_TYPE),
             width, height);
    }

    public PGImage(int [][] pixels){
        this(null, 0, 0);

        if((pixels != null) && (pixels.length > 0)){
            height = pixels.length;
            width = pixels[0].length;
            bi = new BufferedImage(width, height, RGB_TYPE);
            // Get the pixel data to be filled from the buffered image
            DataBufferInt         dbi         =         (DataBufferInt         )
bi.getRaster().getDataBuffer();
            int[] data = dbi.getData();
            // Fill the buffered image with the given pixel values
            for(int i = 0; i < height; i++){
                for(int j = 0; j < width; j++){
                    data[i*width + j] = pixels[i][j];
                }
            }
        }
    }

    public int [][] getAllPixels(){
        int [][] pixels = new int[height][width];

        DataBufferInt dbi = (DataBufferInt ) bi.getRaster().getDataBuffer();
        int[] data = dbi.getData();
        // Retrieve the pixel values
        for(int i = 0; i < height; i++){
            for(int j = 0; j < width; j++){
                pixels[i][j] = data[i*width + j];
            }
        }

        return pixels;
    }

    public int [] getData(){
        return ((DataBufferInt ) bi.getRaster().getDataBuffer()).getData();
    }

    public PGColor getColor(int x, int y){
        int color = getData()[y*width + x];
        int r = (color >> 16) & 0x00FF;
        int g = (color >> 8) & 0x00FF;
        int b = color & 0x00FF;
        return new PGColor(r, g, b);
    }

    public int [][] getPixels(PGRect bound){
        int [][] pixels = new int[bound.height][bound.width];

        DataBufferInt dbi = (DataBufferInt ) bi.getRaster().getDataBuffer();
        int[] data = dbi.getData();
        // Retrieve the pixel values
        for (int i = bound.y; i < bound.y+bound.height; i++) {
            for (int j = bound.x; j < bound.x+bound.width; j++) {
                pixels[i-bound.y][j-bound.x] = data[i * this.width + j];
            }
        }
        return pixels;
    }

    public int getWidth(){
        return width;
    }

    public int getHeight(){
        return height;
    }

    public BufferedImage getBufferedImage(){
        return bi;
    }

    public int getColorType(){
        return type;
    }

    public void setPixel(PGPixel pixel){
        getData()[pixel.y * width + pixel.x] = (pixel.color.a << 24) |
            (pixel.color.r << 16) | (pixel.color.g << 8) | (pixel.color.b);
    }

    /**
     * Deep copy the source image to this image
     * @param src PGImage
     */
    public void copy(PGImage src){
        this.width = src.width;
        this.height = src.height;
        this.type = src.type;
        this.filename = filename;
        this.bi = new BufferedImage(width, height, RGB_TYPE);
        for(int i = 0; i < height * width; i++)
            this.getData()[i] = src.getData()[i];
    }

    public boolean open(String filename, int type){
        if(!(type == RGB_TYPE || type == ARGB_TYPE)){
            System.err.println("Error:  Wrong  type  "  +  type  +  ":  Only
Image.RGY_TYPE " +
                                  "or Image.ARGB_TYPE are acceptable.");
        }
        this.filename = filename;

        try {
            Image i = Toolkit.getDefaultToolkit().getImage(filename);
            MediaTracker mediaTracker = new MediaTracker(new Frame());
            mediaTracker.addImage(i, 0);
            mediaTracker.waitForID(0);

            width = i.getWidth(null);
            height = i.getHeight(null);
            //convert to bufferedImage from image
            bi = new BufferedImage(width, height, type);
            this.type = type;
            Graphics2D g2d = bi.createGraphics();
            g2d.drawImage(i, 0, 0, null);
            return true;
        } catch (Exception e) {
            System.err.println("Error: failed to open " + filename);
            return false;
        }
    }

    public boolean save(){
        String ext = PGUtil.getExtension(filename);
        if (ext.equalsIgnoreCase("jpg") || ext.equalsIgnoreCase("jpeg"))
            return save(JPEG);
        else if(ext.equalsIgnoreCase("png"))
            return save(PNG);
        else
            return save(JPEG);
    }

    private boolean save(int format){
        switch (format) {
        case JPEG:
            try {
                String name = PGUtil.removeExtension(filename);
                ImageIO.write(bi, "jpeg", new File(name + ".jpg"));
            } catch (Exception e) {
                System.err.println("Error: failed to save " + filename +
                                   " in JPEG format");
                return false;
            }
            break;
        case PNG:
            try{
                String name = PGUtil.removeExtension(filename);
                ImageIO.write(bi, "png", new File(name + ".png"));
            } catch (Exception e){
                System.err.println("Error: failed to save " + filename +
                                   " in PNG format");
                return false;
            }
            break;
        default:
            System.err.println("Wrong image format. Only Image.JPEG or " +
                               "Image.PNG are acceptable.");
            return false;
        }

        return true;
    }

    public boolean saveAs(String filename, int format){
        this.filename = filename;
        return save(format);
    }
}
```

11.8 PGFont

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */

import java.awt.Font;

public class PGFont{

    public final static String ARIAL = "Arial";
    public final static String ARIAL_BLACK = "Arial Black";
    public final static String ARIAL_NARROW = "Arial Narrow";
    public final static String BATANG = "Batang";
    public final static String BOOK_ANTIQUA = "Book Antiqua";
    public final static String CENTURY = "Century";
    public final static String CENTURY_GOTHIC = "Century Gothic";
    public final static String COMIC_SANS = "Comic Sans MS";
    public final static String COURIER_NEW = "Courier New";
    public final static String GARAMOND = "Garamond";
    public final static String GEORGIA = "Georgia";
    public final static String GULIM = "Gulim";
    public final static String IMPACT = "Impact";
    public final static String MS_SANS_SERIF = "Microsoft Sans Serif";
    public final static String MONOSPACED = "Monospaced";
    public final static String PAPYRUS = "Papyrus";
    public final static String POOR_RICHARD = "Poor Richard";
    public final static String SANS_SERIF = "SansSerif";
    public final static String SERIF = "Serif";
    public final static String TAHOMA = "Tahoma";
    public final static String TIMES_NEW_ROMAN = "Times New Roman";
    public final static String VERDANA = "Verdana";
    public final static String VRINDA = "Vrinda";
    public final static String WEBDINGS = "Webdings";

    public final static int BOLD = Font.BOLD;
    public final static int ITALIC = Font.ITALIC;
    public final static int PLAIN = Font.PLAIN;

    /** The font family name of this font (see String constants)**/
    public String family;
    /** The type of this font (see integer constants) **/
    public int type;
    /** The size of this font **/
    public int size;

    public PGFont() {
        this(ARIAL, 12, PLAIN);
    }

    public PGFont(String family, int size, int type){
        this.family = family;
        this.size = size;
        this.type = type;
    }
}
```

11.9 PGColor

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGColor{

    /** The RED component of this ARGB color **/
    public int r;
    /** The GREEN component of this ARGB color **/
    public int g;
    /** The BLUE component of this ARGB color **/
    public int b;
    /** The ALPHA(Transparency) component of this ARGB color **/
    public int a;

    /**
     * A default constructor that makes an opaque black color.
     */
    public PGColor() {
        this(0, 0, 0, 255);
    }

    public PGColor(int r, int g, int b, int a){
        if(r > 255)
            this.r = 255;
        else if(r < 0)
            this.r = 0;
        else
            this.r = r;

        if(g > 255)
            this.g = 255;
        else if(g < 0)
            this.g = 0;
        else
            this.g = g;

        if (b > 255)
            this.b = 255;
        else if (b < 0)
            this.b = 0;
        else
            this.b = b;

        if (a > 255)
            this.a = 255;
        else if (a < 0)
            this.a = 0;
        else
            this.a = a;
    }

    public PGColor(int r, int g, int b){
        this(r, g, b, 255);
    }
}
```

11.10 PGLine

```
package photogram;
```

```
/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGLine{

    /** The x-coordinate value of the start point **/
    public int sx;
    /** The y-coordinate value of the start point **/
    public int sy;
    /** The x-coordinate value of the end point **/
    public int ex;
    /** The y-coordinate value of the end point **/
    public int ey;

    public PGLine() {
        this(0, 0, 0, 0);
    }

    public PGLine(int sx, int sy, int ex, int ey){
        this.sx = sx;
        this.sy = sy;
        this.ex = ex;
        this.ey = ey;
    }
}
```

11.11 PGRect

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGRect{

    /** The x-coordinate value of the upper-left corner of this rectangle **/
    public int x;
    /** The y-coordinate value of the upper-left corner of this rectangle **/
    public int y;
    /** The width of this rectangle **/
    public int width;
    /** The height of this rectangle **/
    public int height;

    public PGRect() {
        this(0, 0, 0, 0);
    }

    public PGRect(int x, int y, int width, int height){
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}
```

11.12 PGPixel

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGPixel{

    /** x-coordinate value associated with this pixel **/
    public int x;
    /** y-coordinate value associated with this pixel **/
    public int y;
    /** ARGB color associated with this pixel **/
    public PGColor color;

    public PGPixel() {
        this(new PGColor(), 0, 0);
    }

    public PGPixel(int x, int y){
        this(new PGColor(), x, y);
    }

    public PGPixel(PGColor color){
        this(color, 0, 0);
    }

    public PGPixel(PGColor color, int x, int y){
        this.color = color;
        this.x = x;
        this.y = y;
    }
}
```

11.13 PGOval

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGOval{
    /** The x-coordinate value of the center of this Oval **/
    public int x;
    /** The y-coordinate value of the center of this Oval **/
    public int y;
    /** The length along x-coordinate **/
    public int a;
    /** The length along y-coordinate **/
    public int b;

    public PGOval() {
        this(0, 0, 0, 0);
    }

    public PGOval(int x, int y, int a, int b){
        this.x = x;
        this.y = y;
        this.a = a;
        this.b = b;
    }

}
```

11.14 PGException

```
package photogram;

/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGException extends RuntimeException{
    public PGException(String errMsg) {
        super(errMsg);
    }
}
```

11.15 PGSymbolTable

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.util.*;

public class PGSymbolTable {
    private Stack scope;
    private HashMap varMap;

    public PGSymbolTable() {
        scope = new Stack();
        varMap = new HashMap();
    }

    /**
     * Return the type name of the given variable name
     * @param varName String
     * @return String
     */
    public String get(String varName){
        return (String)varMap.get(varName);
    }

    /**
     * Store a variable with the given type to the current scope stack
     * @param varName String
     * @param typeName String
     */
    public void put(String varName, String typeName){
        varMap.put(varName, typeName);
    }

    public boolean contains(String varName){
        return varMap.containsKey(varName);
    }

    /**
     * Enter its child scope from current scope
     */
    public void enterScope(){
        HashMap currentScope = new HashMap(varMap);
        scope.push(currentScope);
    }

    /**
     * Leave the current scope and go back to the parent scope
     */
    public void leaveScope(){
        varMap = new HashMap((HashMap)scope.pop());
    }

    public HashMap getCurrentScope(){
        return varMap;
    }
}
```

11.16 PGEnvironment

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.util.*;

public class PGEnvironment {
    /** A Symbol Table that contains the id and data type of variables */
    public PGSymbolTable vars;
    /** A hashmap that contains function id as key and PGFunction as value */
    public HashMap funcs;
    /** A hashmap that contains array id as key and PGArray as value */
    public HashMap arrays;
    /** A hashmap that contains constant's id as key and data type as value */
    public HashMap defines;
    /** A hashmap that contains constant's id as key and contant's value as
value.
     * (NOTE: This is required for code generation) */
    public HashMap defines_vals;
    public HashMap defines_class;
    /** A hashmap that contains global variable's id as key and value as value
     * (NOTE: This is required for code generation) */
    //public HashMap global_vals;
    public List globalLines;
    /** An arraylist that contains the name of included files */
    public List includes;
    /** A hashmap that contains variable's id as key and whether it is already
     *  initialized as value (used for generating uninitialized exception) */
    public HashMap var_init_record;
    /** An arraylist that contains the line number where functions are
declared.
     *  (NOTE: This is for making the code generator's job easier and faster)*/
    public List funcLines;
    /** An arraylist that contains the line number where built-in objects are
declared.
     *  (NOTE: This is for making the code generator's job easier and faster)*/
    public List pgTypeLines;

    public HashMap switchLines;
    public HashMap stringCaseLines;

    private Stack arrayScope;
    private Stack initVarScope;

    public HashMap font_const;
    public HashMap image_const;
    public HashMap color_field;
    public HashMap line_field;
    public HashMap oval_field;
    public HashMap pixel_field;
    public HashMap rect_field;

    public HashMap img_obj_funcs;
    public HashMap string_obj_funcs;

    public List lib_func_list;

    public HashMap func_orig_list;

    public List charLines;

    public PGEnvironment() {
        vars = new PGSymbolTable();
        funcs = new HashMap();
        arrays = new HashMap();
        defines = new HashMap();
        defines_vals = new HashMap();
        defines_class = new HashMap();
        globalLines = new ArrayList();
        charLines = new ArrayList();
        includes = new ArrayList();
        var_init_record = new HashMap();
        funcLines = new ArrayList();
        pgTypeLines = new ArrayList();
        switchLines = new HashMap();
        stringCaseLines = new HashMap();

        arrayScope = new Stack();
        initVarScope = new Stack();
        font_const = new HashMap();
        image_const = new HashMap();
        img_obj_funcs = new HashMap();
        string_obj_funcs = new HashMap();
        lib_func_list = new ArrayList();
        func_orig_list = new HashMap();

        color_field = new HashMap();
        oval_field = new HashMap();
        line_field = new HashMap();
        pixel_field = new HashMap();
        rect_field = new HashMap();
    }

    public void arrayEnterScope(){
        HashMap currentArrayScope = new HashMap(arrays);
        arrayScope.push(currentArrayScope);
    }

    public void arrayLeaveScope(){
        arrays = new HashMap((HashMap) arrayScope.pop());
    }

    public void initVarEnterScope(){
        HashMap currentInitVarScope = new HashMap(var_init_record);
        initVarScope.push(currentInitVarScope);
    }

    public void initVarLeaveScope(){
        var_init_record = new HashMap((HashMap) initVarScope.pop());
    }
}
```

11.17 PGReturnTree

```
package photogram;

/**
 * <p>Title: PGReturnTree</p>
 * <p>Description: This class is used to check whether a function has return
 * statement. If the root node has return statement, then it means the function
 * has return statement. Otherwise, it looks into each child nodes to figure
 * out whether all of its child nodes have return statement.</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.util.*;

public class PGReturnTree {

    private Stack parentStack;
    private int currentParentID;
    private HashMap nodeMap;
    private int currentID;
    private int nextID;

    public PGReturnTree() {
        parentStack = new Stack();
        currentParentID = 0;
        nodeMap = new HashMap();
        nextID = 0;
    }

    public void addNode(){
        if(nodeMap.isEmpty()){
            PGReturnNode root = new PGReturnNode(0, -1);
            nodeMap.put(new Integer(root.getID()), root);
            nextID = 1;
        }
        else{
            // Add this node to the tree
            PGReturnNode node = new PGReturnNode(nextID, currentParentID);
            nodeMap.put(new Integer(node.getID()), node);

            // Now get the parent node of this newly added node, and add this
            // new node as its child node
            PGReturnNode     parent     =     (PGReturnNode)     nodeMap.get(new
Integer(currentParentID));
            parent.addChildID(node.getID());

            currentID = nextID;
            nextID++;
        }
    }

    public void enterScope(int scopeLevel){
        if(scopeLevel == 1)
            return;

        parentStack.push(new Integer(currentParentID));
        currentParentID = nextID-1;
    }

    public void leaveScope(int scopeLevel){
        if(scopeLevel == 0)
            return;

        if(!parentStack.isEmpty()){
            currentParentID = ( (Integer) parentStack.pop()).intValue();
            currentID      =      (      (PGReturnNode)      nodeMap.get(new
Integer(currentID))).getParentID();
        }
    }

    /**
     * Sets that the current node has an return statement
     */
    public void setCurrentNodeHasReturn(){
        PGReturnNode node = (PGReturnNode) nodeMap.get(new Integer(currentID));
        node.setHasReturn();
    }

    /**
     * Start from the root node and check whether the root node has an return
     * statement. If not, check whether all of its child has return statement,
     * and repeat this process for all nodes in the tree.
     * @return boolean
     */
    public boolean checkReturnExist(int parent_id){
        // Get the parent node
        PGReturnNode      parent      =      (PGReturnNode)      nodeMap.get(new
Integer(parent_id));
        // If the parent has return statement, then simply return true
        if(parent.hasReturn()){
            return true;
        }
        // Otherwise, check whether all of the childs have return statements
        else{
            // If this is the leaf node (no child), then just return false
            if(parent.getChildIDList().isEmpty()){
                return false;
            }
            // Otherwise, recurse over its children
            else{
                Iterator i = parent.getChildIDList().listIterator();
                boolean childHasReturn = true;
                while(i.hasNext()){
                    childHasReturn                                          &=
checkReturnExist(((Integer)i.next()).intValue());
                }
                return childHasReturn;
            }
        }
    }

    /**
     * This is a helper function for debugging
     */
    public void printTree(){
        TreeMap temp = new TreeMap(nodeMap);
        Iterator i = temp.values().iterator();
        PGReturnNode node = null;
        while(i.hasNext()){
            node = (PGReturnNode) i.next();
            System.out.println(node.toString());
        }
    }

    public void clear(){
        parentStack.clear();
        currentParentID = 0;
        nodeMap.clear();
        nextID = 0;
        currentID = 0;
    }

    public class PGReturnNode {
        private int id;
        private int parentID;
        private List childID;
        private boolean hasReturn;

        public PGReturnNode(int id, int parentID) {
            this.id = id;
            this.parentID = parentID;
            this.childID = new ArrayList();
            hasReturn = false;
        }
        public PGReturnNode(int parentID) {
            this(0, parentID);
        }

        public PGReturnNode() {
            this(0, 0);
        }

        public void setID(int id) {
            this.id = id;
        }

        public void setParentID(int parentID) {
            this.parentID = parentID;
        }

        public void addChildID(int childID) {
            this.childID.add(new Integer(childID));
        }

        public void setHasReturn(){
            this.hasReturn = true;
        }

        public int getID() {
            return id;
        }

        public int getParentID() {
            return parentID;
        }

        public List getChildIDList() {
            return childID;
        }

        public boolean hasReturn(){
            return hasReturn;
        }

        public String toString(){
            String msg = "ID: " + id + ",  Parent: " + parentID + ", Childs: ";
            if(childID.size() == 0)
                msg += "None, ";
            else{
                Iterator i = childID.listIterator();
                while (i.hasNext()) {
                    msg += ""+i.next() + ",";
                }
                msg += " ";
            }
            msg += "HasReturn: " + hasReturn;
            return msg;
        }
    }
}
```

11.18 SemanticAnalyzer

```
package photogram;

/**
```

```
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.util.*;
import java.io.*;

public class SemanticAnalyzer {

    /******************** CONSTANTS FOR THE SIGNS ***********************/

    public static final int MULT = 0;
    public static final int PLUS = 1;
    public static final int MINUS = 2;
    public static final int DIV = 3;
    public static final int MOD = 4;
    public static final int PLUSEQ = 5;
    public static final int MINUSEQ = 6;
    public static final int MULTEQ = 7;
    public static final int DIVEQ = 8;
    public static final int MODEQ = 9;
    public static final int GTE = 10;
    public static final int LTE = 11;
    public static final int GT = 12;
    public static final int LT = 13;
    public static final int EQ = 14;
    public static final int NEQ = 15;
    public static final int NOT = 16;
    public static final int OR = 17;
    public static final int AND = 18;
    public static final int BOR = 19;
    public static final int BAND = 20;
    public static final int BXOR = 21;
    public static final int OREQ = 22;
    public static final int ANDEQ = 23;
    public static final int XOREQ = 24;
    public static final int LSHIFT = 25;
    public static final int RSHIFT = 26;
    public static final int PPLUS = 27;
    public static final int MMINUS = 28;
    public static final int ASGN = 29;

    /********************* CONSTANTS FOR THE TYPES *********************/

    public static final String IntegerType = "int";
    public static final String DoubleType = "double";
    public static final String BooleanType = "boolean";
    public static final String VoidType = "void";
    public static final String ArrayType = "array";
    public static final String FunctionType = "function";
    public static final String ImageType = "Image";
    public static final String PixelType = "Pixel";
    public static final String ColorType = "Color";
    public static final String FontType = "Font";
    public static final String LineType = "Line";
    public static final String StringType = "String";
    public static final String RectType = "Rect";
    public static final String OvalType = "Oval";

    public final String Photogram_Constant = "Photogram_Builtin_Constant";
    public final String Photogram_Field = "Photogram_Builtin_Field";
    public final String Photogram_Array_Length = "Photogram_Array_Length";

    private PGEnvironment env;
    // currentFunction is used to tell what is the current function when using
    // checkReturn method
    private String currentFunction;
    // currentSwitchType is used to check whether the types of cases of current
    // switch statement are consistent
    private Stack currentSwitchType;
    // caseList and currentCaseList is used to find out overlapping cases in a
    // switch statement
    private Stack caseList;
    private List currentCaseList;
    // mainCount is used to check whether there is function call of main, which
    // is not allowed
    private int mainCount;
    // scoreLevel and returnTree is used to check whether function has
    // return statement in all necessary places
    private int scopeLevel;
    private PGReturnTree returnTree;
    // returned is used to find out unreacheable statements, which comes after
    // return statement so they can never be reached in the same scope
    private boolean returned;
    //  inLoopStack are used to check whether break and continue
    // keyword are used outside of loop statement
    private LinkedList isLoopList;
    // inLoopStack are used to check whether break keyword is used
    // outside of switch statement
    private LinkedList isSwitchList;
    // a list of reserved word that are used by Java which cannot be used as
    // define name, var name, or function name
    private List reservedWordList;

    private boolean isLibFunc = false;

    private boolean alreadyHasDefault = false;

    public SemanticAnalyzer() {
        env = new PGEnvironment();
        currentFunction = "";
        currentCaseList = new ArrayList();
        caseList = new Stack();
        currentSwitchType = new Stack();
        returnTree = new PGReturnTree();

        setupReservedWordList();
        setupPGFontConstants();
        setupPGImageConstAndFuncs();
        setupPGStringFunctions();
        setupPGFields();

        mainCount = 0;
        returned = false;
        isLoopList = new LinkedList();
        isSwitchList = new LinkedList();

        loadLibraryFunctions();
    }

    /**
     * Load the library functions
     */
```

```
private void loadLibraryFunctions() {
    PGDataType temp;
    Vector v = new Vector();
    Object[] var_list;

    PGDataType PGDouble = new PGDataType();
    PGDouble.setType("double");
    putLibFunc(new PGFunction(PGDouble, "random", null));
    putLibFunc(new PGFunction(PGDouble, "pi", null));
    putLibFunc(new PGFunction(PGDouble, "e", null));

    PGDataType PGVoid = new PGDataType();
    PGVoid.setType("void");

    PGDataType PGInt = new PGDataType();
    PGInt.setType("int");

    PGDataType PGBoolean = new PGDataType();
    PGBoolean.setType("boolean");

    PGDataType PGString = new PGDataType();
    PGString.setType("String");

    PGDataType PGColor = new PGDataType();
    PGColor.setType("Color");
    putLibFunc(new PGFunction(PGColor, "Color", null));
    v = new Vector();
    v.add(PGInt);
    v.add(PGInt);
    v.add(PGInt);
    var_list = v.toArray();
    putLibFunc(new PGFunction(PGColor, "Color", var_list));

    PGDataType PGImage = new PGDataType();
    PGImage.setType("Image");
    putLibFunc(new PGFunction(PGImage, "Image", null));
    v = new Vector();
    v.add(PGInt);
    v.add(PGInt);
    var_list = v.toArray();
    putLibFunc(new PGFunction(PGImage, "Image", var_list));
    v = new Vector();
    temp = new PGArray(2, "");
    ( (PGArray) temp).setArrayType("int");
    v.add(temp);
    var_list = v.toArray();
    putLibFunc(new PGFunction(PGImage, "Image", var_list));

    PGDataType PGPixel = new PGDataType();
    PGPixel.setType("Pixel");
    putLibFunc(new PGFunction(PGPixel, "Pixel", null));
    v = new Vector();
    v.add(PGInt);
    v.add(PGInt);
    var_list = v.toArray();
    putLibFunc(new PGFunction(PGPixel, "Pixel", var_list));
    v = new Vector();
    v.add(PGColor);
    var_list = v.toArray();
    putLibFunc(new PGFunction(PGPixel, "Pixel", var_list));
    v.add(PGInt);
    v.add(PGInt);
    var_list = v.toArray();
```

```
putLibFunc(new PGFunction(PGPixel, "Pixel", var_list));

PGDataType PGRect = new PGDataType();
PGRect.setType("Rect");
putLibFunc(new PGFunction(PGRect, "Rect", null));
v = new Vector();
v.add(PGInt);
v.add(PGInt);
v.add(PGInt);
v.add(PGInt);
var_list = v.toArray();
putLibFunc(new PGFunction(PGRect, "Rect", var_list));

PGDataType PGOval = new PGDataType();
PGOval.setType("Oval");
putLibFunc(new PGFunction(PGOval, "Oval", null));
putLibFunc(new PGFunction(PGOval, "Oval", var_list));

PGDataType PGLine = new PGDataType();
PGLine.setType("Line");
putLibFunc(new PGFunction(PGLine, "Line", null));
putLibFunc(new PGFunction(PGLine, "Line", var_list));

PGDataType PGFont = new PGDataType();
PGFont.setType("Font");
putLibFunc(new PGFunction(PGFont, "Font", null));
v = new Vector();
v.add(PGString);
v.add(PGInt);
v.add(PGInt);
var_list = v.toArray();
putLibFunc(new PGFunction(PGFont, "Font", var_list));

temp = new PGArray(1, "");
temp.setType("array");
( (PGArray) temp).setArrayType("Image");
v = new Vector();
v.add(PGString);
var_list = v.toArray();
putLibFunc(new PGFunction(temp, "openDir", var_list));

v = new Vector();
v.add(PGDouble);
var_list = v.toArray();
putLibFunc(new PGFunction(PGDouble, "cos", var_list));
putLibFunc(new PGFunction(PGDouble, "sin", var_list));
putLibFunc(new PGFunction(PGDouble, "tan", var_list));
putLibFunc(new PGFunction(PGDouble, "acos", var_list));
putLibFunc(new PGFunction(PGDouble, "asin", var_list));
putLibFunc(new PGFunction(PGDouble, "ceil", var_list));
putLibFunc(new PGFunction(PGDouble, "floor", var_list));
putLibFunc(new PGFunction(PGDouble, "sqrt", var_list));
putLibFunc(new PGFunction(PGDouble, "abs", var_list));
putLibFunc(new PGFunction(PGDouble, "log", var_list));
putLibFunc(new PGFunction(PGDouble, "exp", var_list));
putLibFunc(new PGFunction(PGInt, "doubleToInteger", var_list));

v.add(PGDouble);
var_list = v.toArray();
putLibFunc(new PGFunction(PGDouble, "pow", var_list));

v = new Vector();
v.add(PGInt);
```

```
v.add(PGInt);
var_list = v.toArray();
putLibFunc(new PGFunction(PGDouble, "pow", var_list));
putLibFunc(new PGFunction(PGDouble, "maskBits", var_list));

v = new Vector();
v.add(PGImage);
v.add(PGString);
v.add(PGInt);
v.add(PGInt);
v.add(PGFont);
v.add(PGColor);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "drawText", var_list));

v = new Vector();
v.add(PGImage);
v.add(PGInt);
v.add(PGInt);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "resize", var_list));

v = new Vector();
v.add(PGImage);
v.add(PGImage);
v.add(PGInt);
v.add(PGInt);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "paste", var_list));
//add other paste
v = new Vector();
v.add(PGImage);
v.add(PGImage);
v.add(PGRect);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "paste", var_list));

v = new Vector();
v.add(PGImage);
v.add(PGDouble);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "rotate", var_list));

v = new Vector();
temp = new PGArray(1, "");
temp.setType("array");
( (PGArray) temp).setArrayType("int");
v.add(PGImage);
v.add(temp);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "map", var_list));

v = new Vector();
temp = new PGArray(2, "");
temp.setType("array");
( (PGArray) temp).setArrayType("double");
v.add(PGImage);
v.add(temp);
var_list = v.toArray();
putLibFunc(new PGFunction(PGImage, "convol", var_list));
putLibFunc(new PGFunction(PGImage, "ditherBlock", var_list));

v = new Vector();
```

```
        v.add(PGString);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGVoid, "print", var_list));

        v = new Vector();
        v.add(PGString);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGVoid, "println", var_list));

        v = new Vector();
        v.add(PGImage);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGImage, "grayscale", var_list));
        putLibFunc(new PGFunction(PGImage, "quantUniform", var_list));
        putLibFunc(new PGFunction(PGImage, "quantPopulosity", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherThreshold", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherRandom", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherFS", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherBright", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherOrdered", var_list));
        putLibFunc(new PGFunction(PGImage, "ditherClustered", var_list));
        putLibFunc(new PGFunction(PGImage, "filterBox", var_list));
        putLibFunc(new PGFunction(PGImage, "filterBartlett", var_list));
        putLibFunc(new PGFunction(PGImage, "filterGaussian", var_list));
        putLibFunc(new PGFunction(PGImage, "filterEdgeDetect", var_list));
        putLibFunc(new PGFunction(PGImage, "filterEnhance", var_list));
        putLibFunc(new PGFunction(PGImage, "sizeHalf", var_list));
        putLibFunc(new PGFunction(PGImage, "sizeDouble", var_list));

        v.add(PGInt);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGImage, "filterGaussianN", var_list));

        v = new Vector();
        v.add(PGImage);
        v.add(PGLine);
        v.add(PGColor);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGImage, "drawShape", var_list));
        v = new Vector();
        v.add(PGImage);
        v.add(PGOval);
        v.add(PGColor);
        v.add(PGBoolean);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGImage, "drawShape", var_list));
        v = new Vector();
        v.add(PGImage);
        v.add(PGRect);
        v.add(PGColor);
        v.add(PGBoolean);
        var_list = v.toArray();
        putLibFunc(new PGFunction(PGImage, "drawShape", var_list));
    }

    private void setupPGFields(){
        // Set up the fields for PGColor
        env.color_field.put("r", IntegerType);
        env.color_field.put("g", IntegerType);
        env.color_field.put("b", IntegerType);
        env.color_field.put("a", IntegerType);
        // Set up the fields for PGLine
        env.line_field.put("sx", IntegerType);
        env.line_field.put("sy", IntegerType);
        env.line_field.put("ex", IntegerType);
        env.line_field.put("ey", IntegerType);
        // Set up the fields for PGOval
        env.oval_field.put("x", IntegerType);
        env.oval_field.put("y", IntegerType);
        env.oval_field.put("a", IntegerType);
        env.oval_field.put("b", IntegerType);
        // Set up the fields for PGPixel
        env.pixel_field.put("x", IntegerType);
        env.pixel_field.put("y", IntegerType);
        env.pixel_field.put("color", ColorType);
        // Set up the fields for PGRect
        env.rect_field.put("x", IntegerType);
        env.rect_field.put("y", IntegerType);
        env.rect_field.put("width", IntegerType);
        env.rect_field.put("height", IntegerType);
    }

    private void setupReservedWordList(){
        reservedWordList = new ArrayList();

        reservedWordList.add("abstract");
        reservedWordList.add("do");
        reservedWordList.add("package");
        reservedWordList.add("synchronized");
        reservedWordList.add("this");
        reservedWordList.add("private");
        reservedWordList.add("implements");
        reservedWordList.add("import");
        reservedWordList.add("protected");
        reservedWordList.add("throw");
        reservedWordList.add("throws");
        reservedWordList.add("public");
        reservedWordList.add("instanceof");
        reservedWordList.add("extends");
        reservedWordList.add("byte");
        reservedWordList.add("transient");
        reservedWordList.add("short");
        reservedWordList.add("interface");
        reservedWordList.add("final");
        reservedWordList.add("catch");
        reservedWordList.add("char");
        reservedWordList.add("try");
        reservedWordList.add("long");
        reservedWordList.add("static");
        reservedWordList.add("class");
        reservedWordList.add("float");
        reservedWordList.add("native");
        reservedWordList.add("strictfp");
        reservedWordList.add("volatile");
        reservedWordList.add("super");
        reservedWordList.add("const");
        reservedWordList.add("goto");
        reservedWordList.add("assert");
        reservedWordList.add("PGLibrary");
        reservedWordList.add("PGImage");
        reservedWordList.add("PGFont");
    }

    private void setupPGFontConstants(){
        // Input the PGFont constants
        env.font_const.put("ARIAL", StringType);
```

```
        env.font_const.put("ARIAL_BLACK", StringType);
        env.font_const.put("ARIAL_NARROW", StringType);
        env.font_const.put("BATANG", StringType);
        env.font_const.put("BOOK_ANTIQUA", StringType);
        env.font_const.put("CENTURY", StringType);
        env.font_const.put("CENTURY_GOTHIC", StringType);
        env.font_const.put("COMIC_SANS", StringType);
        env.font_const.put("COURIER_NEW", StringType);
        env.font_const.put("GARAMOND", StringType);
        env.font_const.put("GEORGIA", StringType);
        env.font_const.put("GULIM", StringType);
        env.font_const.put("IMPACT", StringType);
        env.font_const.put("MS_SANS_SERIF", StringType);
        env.font_const.put("MONOSPACED", StringType);
        env.font_const.put("PAPYRUS", StringType);
        env.font_const.put("POOR_RICHARD", StringType);
        env.font_const.put("SANS_SERIF", StringType);
        env.font_const.put("SERIF", StringType);
        env.font_const.put("TAHOMA", StringType);
        env.font_const.put("TIMES_NEW_ROMAN", StringType);
        env.font_const.put("VERDANA", StringType);
        env.font_const.put("VRINDA", StringType);
        env.font_const.put("WEBDINGS", StringType);
        env.font_const.put("BOLD", IntegerType);
        env.font_const.put("ITALIC", IntegerType);
        env.font_const.put("PLAIN", IntegerType);
    }

    private void setupPGImageConstAndFuncs(){
        // Input the PGImage constants
        env.image_const.put("JPEG", IntegerType);
        env.image_const.put("PNG", IntegerType);
        env.image_const.put("RGB_TYPE", IntegerType);
        env.image_const.put("ARGB_TYPE", IntegerType);

        // Input the PGImage object calls
        PGArray array = new PGArray(2, "unknown");
        array.setArrayType(IntegerType);
        env.img_obj_funcs.put("getAllPixels",       new       PGFunction(array,
"getAllPixels", null));

        array = new PGArray(1, "unknown");
        array.setArrayType(IntegerType);
        env.img_obj_funcs.put("getData",   new   PGFunction(array,   "getData",
null));

        PGDataType [] args = new PGDataType[1];
        args[0] = new PGDataType();
        args[0].setType(RectType);
        env.img_obj_funcs.put("getPixels",  new  PGFunction(array,  "getPixels",
args));

        PGDataType data_type = new PGDataType();
        data_type.setType(ColorType);
        args = new PGDataType[2];
        args[0] = new PGDataType();
        args[0].setType(IntegerType);
        args[1] = new PGDataType();
        args[1].setType(IntegerType);
        env.img_obj_funcs.put("getColor", new PGFunction(data_type, "getColor",
args));

        data_type.setType(IntegerType);
        env.img_obj_funcs.put("getWidth", new PGFunction(data_type, "getWidth",
null));
        env.img_obj_funcs.put("getHeight",       new       PGFunction(data_type,
"getHeight", null));
        env.img_obj_funcs.put("getColorType",      new      PGFunction(data_type,
"getColorType", null));

        data_type.setType(VoidType);
        args = new PGDataType[1];
        args[0] = new PGDataType();
        args[0].setType(PixelType);
        env.img_obj_funcs.put("setPixel", new PGFunction(data_type, "setPixel",
args));

        args = new PGDataType[1];
        args[0] = new PGDataType();
        args[0].setType(ImageType);
        env.img_obj_funcs.put("copy", new PGFunction(data_type, "copy", args));

        data_type.setType(BooleanType);
        args = new PGDataType[2];
        args[0] = new PGDataType();
        args[0].setType(StringType);
        args[1] = new PGDataType();
        args[1].setType(IntegerType);
        env.img_obj_funcs.put("open", new PGFunction(data_type, "open", args));
        env.img_obj_funcs.put("save", new PGFunction(data_type, "save", null));
        env.img_obj_funcs.put("saveAs",   new   PGFunction(data_type,   "saveAs",
args));
    }

    private void setupPGStringFunctions(){
        PGDataType returnType = new PGDataType();
        PGDataType [] args = new PGDataType[1];
        args[0] = new PGDataType();
        args[0].setType(IntegerType);
        returnType.setType(StringType);
        env.string_obj_funcs.put("charAt", new PGFunction(returnType, "charAt",
args));

        returnType.setType(IntegerType);
        args[0].setType(StringType);
        env.string_obj_funcs.put("compareTo",     new     PGFunction(returnType,
"compareTo", args));
        env.string_obj_funcs.put("length", new PGFunction(returnType, "length",
null));
        env.string_obj_funcs.put("indexOf",      new      PGFunction(returnType,
"indexOf", args));
        env.string_obj_funcs.put("toLowerCase",    new    PGFunction(returnType,
"toLowerCase", args));
        env.string_obj_funcs.put("toUpperCase",    new    PGFunction(returnType,
"toUpperCase", args));

        returnType.setType(BooleanType);
        env.string_obj_funcs.put("endWith",      new      PGFunction(returnType,
"endWith", args));
        env.string_obj_funcs.put("equals", new PGFunction(returnType, "equals",
args));
        env.string_obj_funcs.put("equalsIgnoreCase", new PGFunction(returnType,
"equalsIgnoreCase", args));
        env.string_obj_funcs.put("startsWith",     new     PGFunction(returnType,
"startsWith", args));
```

```
        env.string_obj_funcs.put("matches",      new      PGFunction(returnType,
"matches", args));

        args = new PGDataType[2];
        args[0] = new PGDataType();
        args[0].setType(StringType);
        args[1] = new PGDataType();
        args[1].setType(StringType);
        returnType.setType(StringType);
        env.string_obj_funcs.put("replaceFirst",    new    PGFunction(returnType,
"replaceFirst", args));
        env.string_obj_funcs.put("replaceAll",     new     PGFunction(returnType,
"replaceAll", args));
    }

    /**
     * Check types for binary operations
     * @param a PGDataType
     * @param b PGDataType
     * @param sign int
     * @throws PGException
     * @return PGDataType
     */
    public  PGDataType  checkType(PGDataType  a,  PGDataType  b,  int  sign)  throws
PGException{
        if(a == null || b == null)
            return null;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, sign);
        PGDataType temp_b = checkTypeException(b, sign);

        PGDataType r = null;
        switch(sign){
            case PLUS:
                r = checkArithOrStringExpression(temp_a, temp_b);
                break;
            case MULT: // fall through
            case MINUS: // fall through
            case DIV: // fall through
            case MOD:
                r = checkArithExpression(temp_a, temp_b, false, sign);
                break;
            case GTE: // fall through
            case LTE: // fall through
            case GT: // fall through
            case LT: // fall through
            case EQ: // fall through
            case NEQ:
                r = checkArithExpression(temp_a, temp_b, true, sign);
                break;
            case OR: // fall through
            case AND:
                r = checkBoolExpression(temp_a, temp_b, sign);
                break;
            case BOR: // fall through
            case BAND: // fall through
            case BXOR: // fall through
            case LSHIFT: // fall through
            case RSHIFT:
                r = checkLogicExpression(temp_a, temp_b, sign);
                break;
        }

        return r;
    }

    /**
     * Check type for unary operations
     * @param a PGDataType
     * @param sign int
     * @throws PGException
     * @return PGDataType
     */
    public PGDataType checkType(PGDataType a, int sign) throws PGException{
        if(a == null)
            return null;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, sign);

        PGDataType r = null;
        PGDataType b = new PGDataType();
        switch(sign){
            case NOT:
                b.setType(BooleanType); // Dummy type for unary expression
                r = checkBoolExpression(temp_a, b, NOT);
                break;
            case PPLUS:
                b.setType(IntegerType); // Dummy type for unary expression
                r = checkArithExpression(temp_a, b, false, PPLUS);
                break;
            case MMINUS:
                b.setType(IntegerType); // Dummy type for unary expression
                r = checkArithExpression(temp_a, b, false, MMINUS);
            case MINUS: // Unary minus
                b.setType(IntegerType);
                r = checkArithExpression(temp_a, b, false, MINUS);
                break;
        }

        return r;
    }

    /**
     * Check type for assign statement
     * @param a PGDataType
     * @param b PGDataType
     * @param lineNum int current line number required for auto type casting
     * during code generation later
     * @param sign int
     * @throws PGException
     */
    public PGDataType assign(PGDataType l_value, PGDataType r_value,
                             boolean isDeclare, int sign, int lineNum) throws
PGException{
        if(l_value == null || r_value == null)
            return null;

        checkUnreacheableStatement();

        String assign_sign = getSign(sign);

        // Make sure that the left value is not a constant
        if(env.defines.containsKey(l_value.getID())){
            throw   new   PGException("Constant   '"+l_value.getID()+"'   is   not
allowed as the " +
                                  "left operand of '"+assign_sign+"'");
        }

        // Check whether []s are used for non-arrays
        if(!env.arrays.containsKey(l_value.getID())   &&   (l_value   instanceof
PGArray)){
            throw new PGException("variable '"+l_value.getID()+"' is not "+
                                  "an array. usage of [] not allowed for " +
                                  "non-arrays");
        }

        // If r_value is one of the built-in object's constructor, then
        // add to pgLines
        if(r_value instanceof PGFunction){
            StringTokenizer st = new StringTokenizer(r_value.getID(), "@");
            if(isBuiltInObject(st.nextToken())  &&  !env.pgTypeLines.contains(new
Integer(lineNum)))
                env.pgTypeLines.add(new Integer(lineNum));
        }

        PGDataType temp_l_value = new PGDataType(l_value.getID());

        if(l_value.getSpecialField()            ==            null            ||
l_value.getSpecialField().equals("")){
            // Check undefined exception for l_value
            checkUndefinedException(l_value);
            temp_l_value.setType(getType(l_value.getID()));
        }
        else if(l_value.getSpecialField().startsWith(Photogram_Constant)){
            throw new PGException("Constant '"+l_value.getSpecialField().
                                  replaceFirst(Photogram_Constant, "")+
                                  "' is not allowed as the " +
                                  "left operand of '"+assign_sign+"'");
        }
        else if(l_value.getSpecialField().startsWith(Photogram_Field)){
            temp_l_value.setType(l_value.getType());
        }
        else if(l_value.getSpecialField().startsWith(Photogram_Array_Length)){
            throw  new  PGException("Assigning  a  value  to  array's  member  field
'length' "+
                                  "not allowed");
        }

        PGDataType r = new PGDataType();

        PGDataType temp_r_value = null;
        PGArray array = null;
        if(sign == ASGN){
            temp_r_value = new PGDataType(r_value.getID());

            if(r_value.getType().equals(FunctionType)){
                // Dimension checking for arrays
                if(((PGFunction)r_value).getReturnType() instanceof PGArray){
                    array = (PGArray)((PGFunction)r_value).getReturnType();
                    if(env.arrays.containsKey(l_value.getID())){
                        PGArray                    l_array                    =
(PGArray)env.arrays.get(l_value.getID());
                        int l_val_dim = 0;
                        if(l_value instanceof PGArray)
                            l_val_dim = ((PGArray)l_value).getDimensions();
                        int l_dim = l_array.getDimensions() - l_val_dim;
                        if (isDeclare)
                            l_dim = l_array.getDimensions();
                        if(l_dim < 0){
                            throw new PGException("Exceeded the array dimension
for array '" +
                                                      l_value.getID() + "': maximum
dimension " +
                                                      l_array.getDimensions()  +  ":
found dimension "
                                                      + l_val_dim);
                        }
                        if(l_dim != array.getDimensions()){
                            String brackets = "";
                            for(int i = 0; i < l_dim; i++)
                                brackets += "[]";
                            throw  new  PGException("Wrong  array  dimension  for
array '" + r_value.getID()
                                                      +    "':    expected    "    +
l_array.getArrayType() + brackets +
                                                      ":       found       "       +
array.toString());
                        }
                    }
                    else{
                        throw   new   PGException("Can   not   assign   "   +
array.toString() +
                                                  " to " + l_value.getType());
                    }

                    temp_r_value.setType(array.getArrayType());
                }
                else{

temp_r_value.setType(((PGFunction)r_value).getReturnType().toString());
                }
            }
            else{
                boolean isArray = false;
                boolean isVariable = false;
                boolean hasID = false;
                if(!r_value.getID().equals("unknown")){
                    checkUndefinedException(r_value);
                    checkUninitializedException(r_value);

                    isArray = env.arrays.containsKey(r_value.getID());
                    isVariable = env.vars.contains(r_value.getID());
                    hasID = true;
                }
                else{
                    isArray = r_value.getType().equals(ArrayType);
                }

                if (isArray) {
                    if (env.arrays.containsKey(l_value.getID())) {
                        PGArray          l_array          =          (PGArray)
env.arrays.get(l_value.getID());
                        int l_val_dim = 0;
                        if (l_value instanceof PGArray)
                            l_val_dim = ( (PGArray) l_value).getDimensions();
                        int l_dim = l_array.getDimensions() - l_val_dim;
                        if (l_dim < 0) {
                            throw new PGException(
                                "Exceeded the array dimension for array '" +
                                l_value.getID() + "': maximum dimension " +
                                l_array.getDimensions() + ": found dimension "
                                + l_val_dim);
                        }
                        int r_dim = 0;
                        PGArray r_array = null;
                        if (hasID) {
                            r_array                  =                 (PGArray)
env.arrays.get(r_value.getID());
                            int r_val_dim = 0;
                            if (r_value instanceof PGArray)
                                r_val_dim         =         (         (PGArray)
r_value).getDimensions();
                            r_dim = r_array.getDimensions() - r_val_dim;
                        }
                        else {
                            r_array = (PGArray) r_value;
                            r_dim = r_array.getDimensions();
                            if (isDeclare)
                                l_dim = ( (PGArray) l_value).getDimensions();
                        }
                        if (r_dim < 0) {
                            throw new PGException(
                                "Exceeded the array dimension for array '" +
                                r_value.getID() + "': maximum dimension " +
                                r_array.getDimensions() + ": found dimension "
                                + ( (PGArray) r_value).getDimensions());
                        }
                        if (l_dim != r_dim) {
                            String r_brackets = "", l_brackets = "";
                            for (int i = 0; i < l_dim; i++)
                                l_brackets += "[]";
                            for (int i = 0; i < r_dim; i++)
                                r_brackets += "[]";
                            throw new PGException("Can not assign " +
                                                  r_array.getArrayType() +
                                                  r_brackets + " to " +
                                                  l_array.getArrayType() +
                                                  l_brackets);
                        }
                        if(isDeclare){

if(!l_array.getArrayType().equals(r_array.getArrayType())){
                                throw   new   PGException("Wrong   data   type   for
array declaration: " +
                                                      "expected       "       +
l_array.getArrayType() +
                                                      ":      found      "      +
r_array.getArrayType());
                            }
                        }
                    }
                    else {
                        if (!hasID) {
                            throw new PGException(
                                "Can  not  assign  using  'new'  keyword  for  non-
array " +
                                "variable '" + l_value.getID() + "'");
                        }
                        else {
                            PGArray         r_array         =         (PGArray)
env.arrays.get(r_value.getID());
```

```
                            int given_r_dim = 0;
                            if(r_value.getType().equals(ArrayType))
                                given_r_dim                                    =
((PGArray)r_value).getDimensions();
                            if (r_array.getDimensions() != given_r_dim) {
                                int   b_count   =   r_array.getDimensions()   -
given_r_dim;
                                if (b_count < 0) {
                                    throw new PGException(
                                        "Exceeded the array dimension for array
'" + r_value.getID() +
                                        "':     maximum     dimension     "     +
r_array.getDimensions() +
                                        ": found dimension " + given_r_dim);
                                }
                                else {
                                    String brackets = "";
                                    for (int i = 0; i < b_count; i++)
                                        brackets += "[]";
                                    throw new PGException(
                                        "Can    not    assign    array    "    +
r_array.getArrayType() +
                                        brackets     +     "     to     "     +
temp_l_value.getType());
                                }
                            }
                        }
                    }

                    if(hasID){
                        temp_r_value.setType(((PGArray)
env.arrays.get(r_value.getID())).getArrayType());
                    }
                    else

temp_r_value.setType(((PGArray)r_value).getArrayType());
                }
                else {
                    if(r_value instanceof PGArray){
                        throw  new  PGException("variable  '"+r_value.getID()+"'
is not "+
                                              "an   array.   usage   of   []   not
allowed for "+
                                              "non-arrays");
                    }

                    if (hasID) {
                        if (isVariable)
                            temp_r_value.setType((String)
env.vars.get(r_value.getID()));
                        else // isConstant
                            temp_r_value.setType((String)
env.defines.get(r_value.getID()));
                    }
                    else
                        temp_r_value.setType(r_value.getType());

                    if(env.arrays.containsKey(l_value.getID())){
                        int given_l_dim = 0;
                        if(l_value.getType().equals(ArrayType)){
                            given_l_dim = ((PGArray)l_value).getDimensions();
                        }
                        PGArray                    l_array                    =
(PGArray)env.arrays.get(l_value.getID());
                        int actual_l_dim = l_array.getDimensions();
                        int dim = actual_l_dim - given_l_dim;
                        if(dim < 0){
                            throw new PGException(
                                "Exceeded the array dimension for array '" +
l_value.getID() +
                                "':      maximum      dimension      "      +
l_array.getDimensions() +
                                ":   found   dimension   "   +   (   (PGArray)
l_value).getDimensions());
                        }
                        else if(dim > 0){
                            String brackets = "";
                            for (int i = 0; i < dim; i++)
                                brackets += "[]";
                            throw new PGException(
                                "Can not assign " + temp_r_value.getType() +
                                "  to  array  "  +  temp_l_value.getType()  +
brackets);
                        }
                    }
                }
            }
        }
        else{
            temp_r_value = checkTypeException(r_value, sign);
        }

        // The l_id is now initialized, so put in the var_init_record list
        // as being initialized
        env.var_init_record.remove(l_value.getID());
        env.var_init_record.put(l_value.getID(), new Boolean(true));

        switch(sign){
            case ASGN:
                r = checkAssign(temp_l_value, temp_r_value);
                break;
            case MULTEQ: // fall through
            case PLUSEQ: // fall through
            case MINUSEQ: // fall through
            case DIVEQ: // fall through
            case MODEQ:
                r     =     checkArithAssignment(temp_l_value,     temp_r_value,
assign_sign);
                break;
            case OREQ: // fall through
            case ANDEQ: // fall through
            case XOREQ: // fall through
                r = checkLogicExpression(temp_l_value, temp_r_value, sign);
                break;
        }

        return r;
    }

    /**
     * Check whether a is defined
     * @param a PGDataType
     */
    private void checkUndefinedException(PGDataType a){
        if            (!            (env.vars.contains(a.getID())            ||
env.defines.containsKey(a.getID())
            || env.arrays.containsKey(a.getID()))) {
            throw new PGException("Variable or Constant '"
                                  + a.getID() + "' not defined.");
        }
    }

    /**
     * Check whether a is initialized at the time of usage
     * @param a PGDataType
     */
    private void checkUninitializedException(PGDataType a){
        // Only check uninitialized exception for variables
        if (env.var_init_record.containsKey(a.getID())) {
            boolean                        initialized                       =
((Boolean)env.var_init_record.get(a.getID())).booleanValue();
            if (!initialized) {
                throw new PGException("Variable '" + a.getID() +
                                      "' not initialized");
            }
        }
    }

    /**
     * Check whether a's type is correct
     * @param a PGDataType
     * @param sign int
     * @return PGDataType
     */
    private PGDataType checkTypeException(PGDataType a, int sign){
        PGDataType temp_a = new PGDataType(a.getID());
        if(a.getType().equals(FunctionType)){
            String          tempName          =          makeFuncName(a.getID(),
((PGFunction)a).getArgTypes(), false);
            PGFunction func = (PGFunction)env.funcs.get(tempName);
            if(func.getReturnType() instanceof PGArray){
                throw      new      PGException("Usage      of      "      +
func.getReturnType().toString()
                                      +  "  not  allowed  for  operator  '"  +
getSign(sign) + "'");
            }

            temp_a.setType(func.getReturnType().toString());
        }
        else{
            // Check undifined and uninitialized exception if a has an 'id'
            if (!a.getID().equals("unknown")) {
                checkUndefinedException(a);
                checkUninitializedException(a);

                // Take care of array dimensions
                if (env.arrays.containsKey(a.getID())) {
                    PGArray array = (PGArray)env.arrays.get(a.getID());
                    int a_dim = 0;
                    if(a instanceof PGArray)
                        a_dim = ((PGArray)a).getDimensions();
                    if(array.getDimensions() != a_dim){
                        int b_count = array.getDimensions() - a_dim;
                        if(b_count < 0){
                            throw new PGException("Exceeded the array dimension
for array '" +
                                                a.getID()   +   "':   maximum
dimension " +
                                                array.getDimensions()   +   ":
found dimension "
                                                + a_dim);
                        }
                        else{
                            String brackets = "";
                            for(int i = 0; i < b_count; i++)
                                brackets += "[]";
                            if(sign >= 0){
                                throw new PGException("Usage of " +
                                    array.getArrayType() +
                                    brackets + " not allowed for operator '" +
                                    getSign(sign) + "'");
                            }
                            else if(sign == -1){
                                throw  new  PGException("boolean  value  expected:
found " +
                                                    array.getArrayType()     +
brackets);
                            }
                            else if(sign == -2){
                                throw  new  PGException("int  or  String  expected:
found " +
                                                    array.getArrayType()     +
brackets);
                            }
                            else if(sign == -3){
                                throw  new  PGException("int  expected  for  array
indexes: found " +
                                                    array.getArrayType()     +
brackets);
                            }
                            else if(sign == -4){
                                throw   new   PGException("Image   expected   for
object calls: found "
                                                    +  array.getArrayType()  +
brackets);
                            }
                        }
                    }

                    temp_a.setType(array.getArrayType());
                }
                else{
                    if(a instanceof PGArray){
                        throw  new  PGException("variable  '"+a.getID()+"'  is  not
an array. "+
                                            "usage of [] not allowed for non-
arrays");
                    }

                    temp_a.setType(getType(a.getID()));
                }
            }
            else{
                temp_a.setType(a.getType());
            }
        }

        return temp_a;
    }
    private PGDataType checkAssign(PGDataType l_value, PGDataType r_value){
        PGDataType r = new PGDataType();

        // We don't allow implicitly assigning double to integer
        if(l_value.getType().equals(IntegerType)                           &&
r_value.getType().equals(DoubleType)){
            throw new PGException("possible loss of precision: assigning " +
                                  "double to integer not allowed");
        }

        if(!(l_value.getType().equals(DoubleType)                          &&
r_value.getType().equals(IntegerType))
           && !l_value.getType().equals(r_value.getType())){
            throw  new  PGException(l_value.getType()  +  "  expected  for  right
value " +
                                  "of     operand     '=':     found     "     +
r_value.getType());
        }

        r.setType(l_value.getType());

        return r;
    }

    private  PGDataType  checkArithAssignment(PGDataType  l_value,  PGDataType
r_value,
                                            String assign_exp){
        PGDataType r = new PGDataType();

        // First check whether both operands a & b are type of integer or
double
        if(!(l_value.getType().equals(IntegerType)                         ||
l_value.getType().equals(DoubleType))){
            throw new PGException("Operator '"+assign_exp+"' can not be applied
" +
                 "to " + l_value.getType() + ": int, or double is required");
        }
        if(!(r_value.getType().equals(IntegerType)                         ||
r_value.getType().equals(DoubleType))){
            throw new PGException("Operator '"+assign_exp+"' can not be applied
" +
                 "to " + r_value.getType() + ": int, or double is required");
        }

        // We don't allow implicitly assigning double to integer
        if(l_value.getType().equals(IntegerType)                           &&
r_value.getType().equals(DoubleType)){
            throw new PGException("possible loss of precision: assigning " +
                                  "double to integer not allowed");
        }

        r.setType(l_value.getType());

        return r;
    }

    private  PGDataType  checkArithOrStringExpression(PGDataType  a,  PGDataType
b){
        PGDataType r = new PGDataType();
        boolean a_exception = false, b_exception = false;
        // If either of the type of a or b is String, then the return type
        // is promoted to String
        if(a.getType().equals(StringType) || b.getType().equals(StringType)){
            if          (!          (a.getType().equals(IntegerType)          ||
a.getType().equals(DoubleType) ||
                   a.getType().equals(StringType)                             ||
a.getType().equals(BooleanType))) {
                a_exception = true;
            }
            if          (!          (b.getType().equals(IntegerType)          ||
b.getType().equals(DoubleType) ||
                   b.getType().equals(StringType)                             ||
b.getType().equals(BooleanType))) {
                b_exception = true;
            }

            r.setType(StringType);
        }
        else           if           (a.getType().equals(DoubleType)           ||
b.getType().equals(DoubleType)) {
            if          (!          (a.getType().equals(IntegerType)          ||
a.getType().equals(DoubleType))) {
                a_exception = true;
            }
            if          (!          (b.getType().equals(IntegerType)          ||
b.getType().equals(DoubleType))) {
                b_exception = true;
            }

            r.setType(DoubleType);
        }
        else {
            if (!a.getType().equals(IntegerType)) {
                a_exception = true;
            }
            if (!b.getType().equals(IntegerType)) {
                b_exception = true;
            }

            r.setType(IntegerType);
        }

        if(a_exception){
            throw new PGException("Operator '+' can not be applied " +
                                  "to " + a.getType() + ": int, double, " +
                                  "or String is required");
        }
        else if(b_exception){
            throw new PGException("Operator '+' can not be applied " +
                                  "to " + b.getType() + ": int, double, " +
                                  "or String is required");
        }

        return r;
    }

    /**
     * Check whether type a and b are valid for arithmatic expression
     * @param a PGDataType
     * @param b PGDataType
     * @return PGDataType
     */
    private PGDataType checkArithExpression(PGDataType a, PGDataType b,
                                            boolean comparison, int sign){
        PGDataType r = new PGDataType();
        String arith_exp = getSign(sign);

        if(!(sign == EQ || sign == NEQ)){
            if(!(a.getType().equals(IntegerType)                              ||
a.getType().equals(DoubleType))){
                throw  new  PGException("Operator  '"+arith_exp+"'  can  not  be
applied " +
                                          "to " + a.getType() + ": int, or double
is required");
            }
            if(!(b.getType().equals(IntegerType)                              ||
b.getType().equals(DoubleType))){
                throw  new  PGException("Operator  '"+arith_exp+"'  can  not  be
applied " +
                                          "to " + b.getType() + ": int, or double
is required");
            }
        }
        else{
            if(!(a.getType().equals(IntegerType)                              ||
a.getType().equals(DoubleType)
                 || a.getType().equals(BooleanType))){
                throw  new  PGException("Operator  '"+arith_exp+"'  can  not  be
applied " +
                                          "to " + a.getType() + ": int, double, or
boolean is required");
            }
            if(!(b.getType().equals(IntegerType)                              ||
b.getType().equals(DoubleType)
                || b.getType().equals(BooleanType))){
                throw  new  PGException("Operator  '"+arith_exp+"'  can  not  be
applied " +
                                          "to " + b.getType() + ": int, double, or
boolean is required");
            }

            if((a.getType().equals(BooleanType)                               &&
!b.getType().equals(BooleanType))
                ||             (b.getType().equals(BooleanType)              &&
!a.getType().equals(BooleanType))){
                 throw  new  PGException("comparing  boolean  value  with  non-
boolean " +
                                           "value not allowed");
             }
        }

        if(comparison){
            r.setType(BooleanType);
        }
        else{
            // If one of the type is double, then the returned type will be
            // promoted to double
            if               (a.getType().equals(DoubleType)                  ||
b.getType().equals(DoubleType)) {
                r.setType(DoubleType);
            }
            else {
                r.setType(IntegerType);
            }
        }

        return r;
```

```
    }

    /**
     * Check whether type a and b are valid for boolean expression
     * @param a PGDataType
     * @param b PGDataType
     * @return PGDataType
     */
    private  PGDataType  checkBoolExpression(PGDataType  a,  PGDataType  b,  int
sign){
        PGDataType r = new PGDataType();

        String bool_exp = getSign(sign);

        if(!a.getType().equals(BooleanType)){
            throw new PGException("Operator '"+bool_exp+"' can not be applied "
+
                     "to " + a.getType() + ": boolean is required");
        }
        if(!b.getType().equals(BooleanType)){
            throw new PGException("Operator '"+bool_exp+"' can not be applied "
+
                     "to " + b.getType() + ": boolean is required");
        }

        r.setType(BooleanType);

        return r;
    }

    /**
     * Check whether type a and b are valid for logical operations
     * @param a PGDataType
     * @param b PGDataType
     * @return PGDataType
     */
    private  PGDataType  checkLogicExpression(PGDataType  a,  PGDataType  b,  int
sign){
        PGDataType r = new PGDataType();

        String logic_exp = "";
        switch(sign){
            case RSHIFT: logic_exp = ">>"; break;
            case LSHIFT: logic_exp = "<<"; break;
            case BAND: logic_exp = "&"; break;
            case BOR: logic_exp = "|"; break;
            case BXOR: logic_exp = "^"; break;
            case OREQ: logic_exp = "|="; break;
            case ANDEQ: logic_exp = "&="; break;
            case XOREQ: logic_exp = "^"; break;
        }

        if(!(a.getType().equals(IntegerType)                                  ||
a.getType().equals(DoubleType))){
            throw  new  PGException("Operator  '"+logic_exp+"'  can  not  be  applied
" +
                     "to  "  +  a.getType()  +  "  as  left  operand:  int,  or  double  is
required");
        }
        if(!b.getType().equals(IntegerType)){
            throw  new  PGException("Operator  '"+logic_exp+"'  can  not  be  applied
" +
                     "to " + b.getType() + " as right operand: int is required");
        }

        r.setType(a.getType());

        return r;
    }

    /**
     * Put this variable 'varName' into the symbol table for variables. <br>
     * @param varName String
     * @param varType String
     * @throws PGException
     */
    public void putVar(PGDataType a, boolean isGlobal, boolean isFuncArg,
                       int lineNum) throws PGException{
        if(a == null)
            return;

        checkReservedWord(a.getID(), "variable");

        if(env.vars.contains(a.getID())  ||  env.defines.containsKey(a.getID())
||
           env.arrays.containsKey(a.getID())){
            throw new PGException("Variable: '" + a.getID() +
                "' is already defined. Redefinition of variable not allowed.");
        }

        if(isFuncArg){
            // Function arguments should be initialized
            env.var_init_record.put(a.getID(), new Boolean(true));
        }
        else{
            // Initialy, the variable is not initialized yet
            env.var_init_record.put(a.getID(), new Boolean(false));
        }

        String data_type = a.getType();
        if(data_type.equals(ArrayType)){
            data_type = ((PGArray)a).getArrayType();
        }
        if(isBuiltInObject(data_type)      &&      !env.pgTypeLines.contains(new
Integer(lineNum)))
            env.pgTypeLines.add(new Integer(lineNum));

        if(a.getType().equals(ArrayType))
            env.arrays.put(a.getID(), a);
        else
            env.vars.put(a.getID(), a.getType());

        if(isGlobal) env.globalLines.add(new Integer(lineNum));
    }

    private String getType(String id){
        String data_type = "";

        if(env.vars.contains(id)){
            data_type = (String) env.vars.get(id);
        }
        else if(env.defines.containsKey(id)){
            data_type = (String) env.defines.get(id);
        }
        else if(env.arrays.containsKey(id)){
            data_type = ((PGArray)env.arrays.get(id)).getArrayType();
        }
        else{ // function
            PGFunction function = null;
            if(env.funcs.containsKey(id)){
                function = (PGFunction)env.funcs.get(id);
            }
            else if(env.img_obj_funcs.containsKey(id)){
                function = (PGFunction)env.img_obj_funcs.get(id);
            }
            else if(env.string_obj_funcs.containsKey(id)){
                function = (PGFunction)env.string_obj_funcs.get(id);
            }

            if(function.getReturnType() instanceof PGArray){
                data_type = ((PGArray)function.getReturnType()).getArrayType();
            }
            else
                data_type = function.getReturnType().toString();
        }

        return data_type;
    }

    public static boolean isBuiltInObject(String type){
        return (type.equals(ImageType) || type.equals(ColorType) ||
                type.equals(FontType) || type.equals(PixelType) ||
                type.equals(OvalType) ||
                type.equals(LineType) || type.equals(RectType));
    }

    public void putLibFunc(PGDataType a) throws PGException{
        if(!isBuiltInObject(a.getID())){
            env.lib_func_list.add(a.getID());
            isLibFunc = true;
        }
        putFunc(a, -1);
    }

    public void putFunc(PGDataType a, int lineNum) throws PGException{
        if(a == null)
            return;

        checkReservedWord(a.getID(), "function");

        // We do not allow function that has the same name as the Library
        // function, even if it's a overloaded function
        if(!isLibFunc){
            if(env.lib_func_list.contains(a.getID())){
                throw  new  PGException("Function  name  '"+a.getID()+"'  already
exists "
                                    +  "as  Photogram  library  function.  Using
the "
                                    + "same name as library function name not
allowed "
                                    + "even if the arguments are different");
            }
        }
        isLibFunc = false;

        PGFunction func = (PGFunction)a;

        // Also, we don't allow function overloading across different files
        if (env.func_orig_list.containsKey(func.getID())) {
            if
(!func.getClassName().equals((String)env.func_orig_list.get(func.getID()))) {
                throw new PGException("Function name '" + func.getID() + "'
already exists "
                                      +    "in    included    file    '"    +
(String)env.func_orig_list.get(func.getID())
                                      + "'. Overloading function across files
not "
                                      + "allowed");
            }
        }

        if(func.getID().equals("main")){
            if(env.funcs.containsKey("main@")                                 ||
env.funcs.containsKey("main@_String[]")){
                throw new PGException("main method already defined");
            }

            if (!func.getReturnType().toString().equals(VoidType)) {
                throw new PGException("main method is required to to have
return " +
                                      "type of void: found " +
                                      func.getReturnType().toString());
            }

            if(func.getArgTypes() != null){
                if (func.getArgTypes().length > 1) {
                    throw new PGException("main method can only have no
arguments or " +
                                          "one argument of type String []");
                }
                else if (func.getArgTypes().length == 1) {
                    if (!func.getArgTypes()[0].toString().equals("String[]")) {
                        throw new PGException("main method can only have
argument type "
                                              + "of String []: found " +

func.getArgTypes()[0].toString());
                    }
                }
            }
        }

        // Return type of void arrays not allowed
        if(func.getReturnType().getType().equals(ArrayType) &&
           ((PGArray)func.getReturnType()).getArrayType().equals(VoidType)){
            throw new PGException("return type of void arrays not allowed");
        }

        String tempName = makeFuncName(a.getID(), func.getArgTypes(), true);

        if(env.funcs.containsKey(tempName)){
            String errMsg = "Function '" + a.getID() + "(";
            if(func.getArgTypes() != null){
                for (int i = 0; i < func.getArgTypes().length - 1; i++) {
                    errMsg += func.getArgTypes()[i].toString() + ", ";
                }
                errMsg    +=    func.getArgTypes()[func.getArgTypes().length    -
1].toString();
            }
            errMsg += ")' is already defined";
            throw new PGException(errMsg);
        }
        env.funcs.put(tempName, a);

        if(lineNum < 0)
            return;

        env.funcLines.add(new Integer(lineNum));

        // Get argument list and check whether built-in objects are used
        // so that can add to pgTypeLines list
        Object [] dataTypes = func.getArgTypes();
        String data_type = "";
        if(dataTypes != null){
            for (int i = 0; i < dataTypes.length; i++) {
                if(dataTypes[i] instanceof PGArray){
                    data_type = ((PGArray)dataTypes[i]).getArrayType();
                }
                else if(dataTypes[i] instanceof PGDataType){
                    data_type = ((PGDataType)dataTypes[i]).getType();
                }
                // If the data type of an argument is PG data type
                if (! (data_type.equals(IntegerType) ||
                       data_type.equals(DoubleType)
                       || data_type.equals(BooleanType))) {
                    env.pgTypeLines.add(new Integer(lineNum));
                    break;
                }
            }
        }

        if(!env.func_orig_list.containsKey(func.getID()))
            env.func_orig_list.put(func.getID(), func.getClassName());
    }

    /**
     * Get the return type of function 'funcName'
     * @param funcName String
     * @throws PGException
     * @return PGDataType
     */
    public  PGDataType  getFunc(String  funcName,  Object  []  arg_ids,  boolean
isEnterScope)
        throws PGException{

        if(funcName == null)
            return null;

        checkUnreacheableStatement();

        if(arg_ids != null){
            for(int i = 0; i < arg_ids.length; i++){
                if(arg_ids[i] == null)
                    return null;
            }
        }

        PGFunction function = null;
        String tempName = makeFuncName(funcName, arg_ids, isEnterScope);
        if(tempName.startsWith("main@")){
            mainCount++;
            if(mainCount > 1){
                throw new PGException("Calling main function not allowed");
            }
        }
        // First check whether this function exists
        if(!(env.funcs.containsKey(tempName)                                  ||
env.img_obj_funcs.containsKey(funcName)
             || env.string_obj_funcs.containsKey(funcName))){
            String errMsg = "Function '" + funcName + "(";
            if(arg_ids != null){
                for (int i = 0; i < arg_ids.length; i++) {
                    PGDataType a = (PGDataType) arg_ids[i];
                    if(a.getType().equals(FunctionType)){
                        tempName            =            makeFuncName(a.getID(),
((PGFunction)a).getArgTypes(),true);
                        PGFunction func = (PGFunction) env.funcs.get(tempName);
                        errMsg += func.getReturnType().toString();
                    }
                    else{
                        if (!a.getID().equals("unknown")) {
                            if(env.arrays.containsKey(a.getID())){
                                int given_dim = 0;
                                if (a instanceof PGArray) {
                                    given_dim = ( (PGArray) a).getDimensions();
                                }
                                PGArray         array         =         (PGArray)
env.arrays.get(a.getID());
                                int dim = array.getDimensions() - given_dim;
                                errMsg += array.getArrayType();
                                for (int j = 0; j < dim; j++)
                                    errMsg += "[]";
                            }
                            else{
                                errMsg += getType(a.getID());
                            }
                        }
                        else {
                            errMsg += a.getType();
                        }
                    }

                    if(i != arg_ids.length -1){
                        errMsg += ", ";
                    }
                }
            }
            errMsg += ")' is not defined";
            throw new PGException(errMsg);
        }

        if(env.funcs.containsKey(tempName)){
            function = (PGFunction) env.funcs.get(tempName);
        }
        else if(env.img_obj_funcs.containsKey(funcName)){
            function = (PGFunction) env.img_obj_funcs.get(funcName);
        }
        else{ // String object function call
            function = (PGFunction) env.string_obj_funcs.get(funcName);
        }

        if(function.getArgTypes() != null){
            // Now check whether there are correct number of arguments for this
            // function, and whether the data types are correct
            int args_length = 0;
            if(arg_ids != null)
                args_length = arg_ids.length;
            if (function.getArgTypes().length != args_length) {
                throw new PGException(
                    "Wrong number of arguments for function '" +
                    funcName + "'");
            }

            String data_type = "";
            PGDataType a = null;
            for (int i = 0; i < args_length; i++) {
                a = (PGDataType) arg_ids[i];
                String brackets = "";
                if(env.funcs.containsKey(tempName)){
                    data_type                                           =
((PGFunction)env.funcs.get(tempName)).getArgTypes()[i].toString();
                }
                /*else if(env.img_obj_funcs.containsKey(funcName)){
                    data_type                                           =
((PGFunction)env.img_obj_funcs.get(funcName)).getArgTypes()[i].toString();
                }*/
                else{
                    // First make sure the ids are defined and initialized
                    if (!a.getID().equals("unknown")) {
                        if(a instanceof PGFunction){
                            tempName          =          makeFuncName(a.getID(),
((PGFunction)a).getArgTypes(),false);
                            PGFunction        func        =        (PGFunction)
env.funcs.get(tempName);
                            data_type = func.getReturnType().toString();
                        }
                        else{
                            checkUndefinedException(a);
                            checkUninitializedException(a);

                            if (env.arrays.containsKey(a.getID())) {
                                data_type         =         (          (PGArray)
env.arrays.get(a.getID())).getArrayType();
                                int a_dim = 0;
                                if (a instanceof PGArray)
                                    a_dim = ( (PGArray) a).getDimensions();
                                int      num_bracket      =      (      (PGArray)
env.arrays.get(a.getID())).
                                    getDimensions() - a_dim;
                                if (num_bracket < 0) {
                                    throw  new  PGException("Exceeded  the  array
dimension for array '" +
                                        a.getID() + "': maximum dimension " +
                                        (                               (PGArray)
env.arrays.get(a.getID())).getDimensions() +
                                        ": found dimension " + a_dim);
                                }
                                for (int j = 0; j < num_bracket; j++)
                                    brackets += "[]";
                                data_type += brackets;
                            }
                            else {
                                if (env.vars.contains(a.getID()))
                                    data_type            =            (String)
env.vars.get(a.getID());
                                else
                                    data_type            =            (String)
env.defines.get(a.getID());
                            }
                        }
                    }
                    else {
                        data_type = a.getType();
                    }
                }

                if (!data_type.equals(function.getArgTypes()[i].toString()) &&
                    ! (data_type.equals(IntegerType) &&

function.getArgTypes()[i].toString().equals(DoubleType))) {
                    String errMsg = "Wrong data type";
                    if(!a.getID().equals("unknown")){
                        errMsg += " for variable or function '" + a.getID() +
"'";
                    }
                    errMsg += ": expected " + function.getArgTypes()[i] +
                                        ", but found " + data_type;
                    throw new PGException(errMsg);
                }
            }
        }
        else{
            if(arg_ids != null){
                throw new PGException("Wrong number of arguments for function
'"
                                        + funcName + "'");
            }
        }

        return function;
    }

    private String makeFuncName(String funcName, Object [] args, boolean
isPutFunc){
        String newFuncName = funcName + "@";
        if (args != null) {
            PGDataType a = null;
            for (int i = 0; i < args.length; i++) {
                a = (PGDataType) args[i];

                if(!a.getID().equals("unknown")){
                    if(isPutFunc){
                        newFuncName += "_" + a.toString();
                    }
                    else{
                        if(a instanceof PGFunction){
                            String tempName = makeFuncName(a.getID(),
((PGFunction)a).getArgTypes(),true);
                            PGFunction func = (PGFunction)
env.funcs.get(tempName);
                            newFuncName += "_" +
func.getReturnType().toString();
                        }
                        else{
                            checkUndefinedException(a);
                            checkUninitializedException(a);

                            if (env.arrays.containsKey(a.getID())) {
                                int given_dim = 0;
                                if (a instanceof PGArray) {
                                    given_dim = ( (PGArray) a).getDimensions();
                                }
                                PGArray        array        =        (PGArray)
env.arrays.get(a.getID());
                                int dim = array.getDimensions() - given_dim;
                                if (dim < 0) {
                                    throw  new  PGException("Exceeded  the  array
dimension for array '" +
                                        a.getID() + "': maximum dimension " +
                                        array.getDimensions()    +    ":    found
dimension "
                                        + given_dim);
                                }
                                newFuncName += "_" + array.getArrayType();
                                for(int j = 0; j < dim; j++)
                                    newFuncName += "[]";
                            }
                            else {
                                if (a instanceof PGArray) {
                                    throw   new   PGException("variable   '"   +
a.getID() + "' is not an array. " +
                                        "usage  of  []  not  allowed  for  non-
arrays");
                                }

                                newFuncName += "_" + getType(a.getID());
                            }
                        }
                    }
                }
                else{
                    newFuncName += "_" + a.toString();
                }
            }
        }
        return newFuncName;
    }

    /**
     * Put this constant 'id' into the symbol table for variables. <br>
     * @param id String
     * @param type String
     */
    public void putDefine(PGDataType a, String value, File className) throws
PGException{
        checkReservedWord(a.getID(), "constant");
        if(env.vars.contains(a.getID()) || env.defines.containsKey(a.getID())
           || env.arrays.containsKey(a.getID())){
            throw new PGException("Constant: " + a.getID() +
                " is already defined. Redefinition of constant not allowed.");
        }

        // Constants and Variables share the same namespace, but constants
        // are put in different place so that it can be differentiated from
        // variables (constants can not be modified, while variables can)
        env.defines.put(a.getID(), a.getType());
        env.defines_vals.put(a.getID(), value);
        env.defines_class.put(a.getID(), className.getName());
    }

    public void putInclude(String id) throws PGException{
        if(env.includes.contains(id)){
            throw  new  PGException("Include  file  '"  +  id  +  "'  is  already
included");
        }
        env.includes.add(id);
    }

    public void checkReturn(PGDataType a) throws PGException{
        if(a == null || currentFunction.equals(""))
            return;

        checkUnreacheableStatement();

        String data_type = "";
        if(a.getType().equals(FunctionType)){
            String          tempName          =          makeFuncName(a.getID(),
((PGFunction)a).getArgTypes(), false);
            data_type                                                          =
((PGFunction)env.funcs.get(tempName)).getReturnType().toString();
        }
        else{
            if (!a.getID().equals("unknown")) {
                checkUndefinedException(a);
                checkUninitializedException(a);

                String brackets = "";
                if (env.arrays.containsKey(a.getID())) {
                    data_type             =             (              (PGArray)
env.arrays.get(a.getID())).getArrayType();
                    int a_dim = 0;
                    if (a instanceof PGArray)
                        a_dim = ( (PGArray) a).getDimensions();
                    int        num_bracket        =        (         (PGArray)
env.arrays.get(a.getID())).getDimensions() - a_dim;
                    if (num_bracket < 0) {
                        throw new PGException("Exceeded the array dimension for
array '" +
                                              a.getID() + "': maximum dimension
" +
                                              (                        (PGArray)
env.arrays.get(a.getID())).getDimensions() +
                                              ": found dimension " + a_dim);
                    }
                    for (int j = 0; j < num_bracket; j++)
                        brackets += "[]";
                    data_type += brackets;
                }
                else if (env.vars.contains(a.getID()))
                    data_type = (String) env.vars.get(a.getID());
                else
                    data_type = (String) env.defines.get(a.getID());
            }
            else {
                data_type = a.getType();
            }
        }

        PGFunction func = (PGFunction) env.funcs.get(currentFunction);

        if (!data_type.equals(func.getReturnType().toString())) {
            String errMsg = "Wrong return type for function '";
            StringTokenizer st = new StringTokenizer(currentFunction, "@");
            errMsg += st.nextToken() + "(";
            if(st.hasMoreTokens()){
                st = new StringTokenizer(st.nextToken(), "_");
                errMsg += st.nextToken();
                while(st.hasMoreTokens()){
                    errMsg += ", " + st.nextToken();
                }
            }
            errMsg  +=  ")':  expected  "  +  func.getReturnType()  +  ":  found  "  +
data_type;
            throw new PGException(errMsg);
        }

        returnTree.setCurrentNodeHasReturn();
        returned = true;
    }

    public PGDataType checkConst(PGDataType a, int lineNum) throws PGException{
        if(a == null)
            return null;

        checkUnreacheableStatement();

        PGDataType r = new PGDataType();

        if(a.getType().equals(ImageType)                                &&
env.image_const.containsKey(a.getID())){
            r.setType( (String) env.image_const.get(a.getID()));
        }
        else                 if(a.getType().equals(FontType)                 &&
env.font_const.containsKey(a.getID())){
            r.setType( (String) env.font_const.get(a.getID()));
        }
        else{
            throw new PGException("Constant '" + a.getType() + "." + a.getID()
                                  + "' does not exist");
        }

        if (!env.pgTypeLines.contains(new Integer(lineNum)))
            env.pgTypeLines.add(new Integer(lineNum));

        r.setSpecialField(Photogram_Constant + a.getType() + "." + a.getID());

        return r;
    }

    public  PGDataType  checkConst(PGDataType  a,  PGDataType  b,  int  lineNum)
throws PGException{
        if(a == null || b == null)
            return null;

        checkUnreacheableStatement();

        checkUndefinedException(a);
        checkUninitializedException(a);

        PGDataType r = new PGDataType();

        String data_type = "";
        if (env.arrays.containsKey(a.getID())) {
            int given_dim = 0;
            if (a instanceof PGArray) {
                given_dim = ( (PGArray) a).getDimensions();
            }
            PGArray array = (PGArray) env.arrays.get(a.getID());
            int dim = array.getDimensions() - given_dim;
            if (dim < 0) {
                throw new PGException("Exceeded the array dimension for array
'" +
                                      a.getID() + "': maximum dimension " +
                                      array.getDimensions()    +    ":    found
dimension "
                                      + given_dim);

            }
            else if (dim > 0) { // Now we're sure it's an array
                if (!b.getID().equals("length")) {
                    throw new PGException("Object call '" + b.getID() +
                                          "' not allowed for arrays. The only
allowed " +
                                          "object    call    for    arrays    is
'length'");
                }

                r.setSpecialField(Photogram_Array_Length);
                r.setType(IntegerType);
                return r;
            }
            else{ // dim == 0
                data_type = array.getArrayType();
            }
        }
        else {
            if (a instanceof PGArray) {
                throw new PGException("variable '" + a.getID() + "' is not an
array. " +
                                      "usage  of  []  not  allowed  for  non-
arrays");
            }

            data_type = (String) env.vars.get(a.getID());
        }

        if(!isBuiltInObject(data_type)){
            throw  new  PGException("Built-in  object  expected  for  accessing
member varaibles: "
                                  + "found " + data_type);
        }

        if(data_type.equals(ImageType)                                    &&
env.image_const.containsKey(b.getID()))
            r.setType((String)env.image_const.get(b.getID()));
        else                 if(data_type.equals(FontType)                &&
env.font_const.containsKey(b.getID()))
            r.setType((String)env.font_const.get(b.getID()));
        else                if(data_type.equals(ColorType)                &&
env.color_field.containsKey(b.getID()))
            r.setType((String)env.color_field.get(b.getID()));
        else                 if(data_type.equals(OvalType)                &&
env.oval_field.containsKey(b.getID()))
            r.setType((String)env.oval_field.get(b.getID()));
        else                 if(data_type.equals(LineType)                &&
env.line_field.containsKey(b.getID()))
            r.setType((String)env.line_field.get(b.getID()));
        else                if(data_type.equals(PixelType)                &&
env.pixel_field.containsKey(b.getID()))
            r.setType((String)env.pixel_field.get(b.getID()));
        else                 if(data_type.equals(RectType)                &&
env.rect_field.containsKey(b.getID()))
            r.setType((String)env.rect_field.get(b.getID()));
        else{
            throw new PGException("Constant or Variable '" + data_type + "." +
b.getID()
                                    + "' does not exist");
        }

        r.setSpecialField(Photogram_Field + data_type + "." + b.getID());
        return r;
    }

    public PGDataType checkObjectCall(PGDataType a, PGDataType func,
                                    Object   []   args,   int   lineNum)   throws
PGException{
        if(a == null || func == null)
            return null;

        checkUnreacheableStatement();

        PGDataType r = new PGDataType();

        PGDataType temp_a = checkTypeException(a, -4);

        if(!(temp_a.getType().equals(ImageType)                              ||
temp_a.getType().equals(StringType))){
            throw  new  PGException("Wrong  data  type  for  object  call:  expected
Image or String: "
                                    + "found " + temp_a.getType());
        }

        getFunc(func.getID(), args, false);

        String temp = "";
        if(temp_a.getType().equals(ImageType)){
            if (env.img_obj_funcs.containsKey(func.getID()))
                temp              =              (              (PGFunction)
env.img_obj_funcs.get(func.getID())).getReturnType().toString();
            else{
                String errMsg = "Function '" + func.getID() + "(";
                if (args != null) {
                    for (int i = 0; i < args.length - 1; i++) {
                        errMsg += args[i].toString() + ", ";
                    }
                    errMsg += args[args.length - 1].toString();
                }
                errMsg += ")' is not defined as Image object call";
                throw new PGException(errMsg);
            }
        }
        else{ // String Type
            if (env.string_obj_funcs.containsKey(func.getID())) {
                temp              =              (              (PGFunction)
env.string_obj_funcs.get(func.getID())).getReturnType().toString();
                if(func.getID().equals("charAt")  &&  !env.charLines.contains(new
Integer(lineNum))){
                    env.charLines.add(new Integer(lineNum));
                }
            }
            else{
                String errMsg = "Function '" + func.getID() + "(";
                if (args != null) {
                    for (int i = 0; i < args.length - 1; i++) {
                        errMsg += args[i].toString() + ", ";
                    }
                    errMsg += args[args.length - 1].toString();
                }
                errMsg += ")' is not defined as String object call";
                throw new PGException(errMsg);
            }

        }

        r.setType(temp);

        return r;
    }

    public void checkFor(PGDataType a) throws PGException{
        if(a == null)
            return;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, -1);
        if(!temp_a.getType().equals(BooleanType)){
            throw new PGException("boolean value expected for 2nd condition of 
"
                                  + "for loop: found " + temp_a.getType());
        }
    }

    public void checkWhile(PGDataType a) throws PGException{
        if(a == null)
            return;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, -1);
        if(!temp_a.getType().equals(BooleanType)){
            throw new PGException("boolean value expected for while condition: 
"
                                  + "found " + temp_a.getType());
        }
    }

    public void checkIf(PGDataType a) throws PGException{
        if(a == null)
            return;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, -1);
        if(!temp_a.getType().equals(BooleanType)){
            throw new PGException("boolean value expected for if condition: "
                                  + "found " + temp_a.getType());
        }
    }

    public void checkSwitch(PGDataType a, int lineNum) throws PGException{
        if(a == null)
            return;

        checkUnreacheableStatement();

        PGDataType temp_a = checkTypeException(a, -2);
        currentSwitchType.push(temp_a.getType());
        // Push the current case list to the stack for taking care of nested
        // switch statement
        caseList.push(currentCaseList);
        currentCaseList = new ArrayList();
        if(!(temp_a.getType().equals(IntegerType)                            ||
temp_a.getType().equals(StringType))){
            throw new PGException("int or String expected for switch statement:
"
                                  + "found " + temp_a.getType());
        }

        // If the switch variable is a String type, then we need to conver it
        // to char type at the time of code generation since Java doesn't
        // allow String type for switch statement
        if(temp_a.getType().equals(StringType)                               &&
!env.switchLines.containsKey(new Integer(lineNum))){
            env.switchLines.put(new Integer(lineNum), a.getID());
        }


    }

    public void checkBreak() throws PGException {
        // In order to tell whether the break statement is in the switch or
        // or loop statement, we need to check whether it or it's ancestors
        // scope is for or switch statement
        boolean isInLoop = false;
        Object [] temp = isLoopList.toArray();
        for(int i = isLoopList.size()-1; i >= 0; i--){
            if(((Boolean)isLoopList.get(i)).booleanValue()){
                isInLoop = true;
                break;
            }
        }
        if(!isInLoop){
            for (int i = isSwitchList.size() - 1; i >= 0; i--) {
                if ( ( (Boolean) isSwitchList.get(i)).booleanValue()) {
                    isInLoop = true;
                    break;
                }
            }
        }
        if (!isInLoop) {
            throw new PGException("break outside of switch or loop statement");
        }

        // break will also cause unreacheable exception
        returned = true;
    }

    public void checkContinue() throws PGException{
        boolean isInLoop = false;
        for(int i = isLoopList.size()-1; i >= 0; i--){
            if(((Boolean)isLoopList.get(i)).booleanValue()){
                isInLoop = true;
                break;
            }
        }

        if(!isInLoop){
            throw new PGException("continue outside of loop statement");
        }

        // continue will also cause unreacheable exception
        returned = true;
    }

    /**
     * Check whether the array index is an integer
     * @param a PGDataType
     * @throws PGException
     */
    public void checkArrayIndex(PGDataType a) throws PGException{
        if(a == null)
            return;

        PGDataType temp_a = checkTypeException(a, -3);
        if(!(temp_a.getType().equals(IntegerType))){
            throw new PGException("int expected for array indexes: "
                                  + "found " + temp_a.getType());
        }
    }

    /**
     * Check whether there are any duplicated cases in one switch statement,
and
     * also check whether the data type of the cases are consistent with the
     * switch variable, and make sure the type is either String with length of
     * 1 or an integer
     * @param a PGDataType
     * @param lineNum int
     * @throws PGException
     */
    public void putCase(PGDataType a, int lineNum) throws PGException{
        returned = false;

        // Check whether this case is already mentioned
        if(currentCaseList.contains(a.getID())){
            throw new PGException("case " + a.getID() + " is already defined");
        }

        // Check whether the datatype is correct for current switch scope
        if(!currentSwitchType.isEmpty() &&
           !a.getType().equals((String)currentSwitchType.peek())){
            throw new PGException("case " + a.getID() + " has wrong data type:
"
                                  + "expected " + currentSwitchType.peek() +
                                  ": found " + a.getType());
        }

        // If the type is String, then only allows String of length 1
        if(a.getType().equals(StringType)){
            if(a.getID().length() != 1){
                throw new PGException("only String with 1 character is allowed
for "
                                      + "case statement: found '" + a.getID() +
"'");
            }
            // Need to convert String cases to char cases in the Java code
            else if(!env.stringCaseLines.containsKey(new Integer(lineNum))){ //
length == 1
                env.stringCaseLines.put(new Integer(lineNum), a.getID());
            }
        }

        currentCaseList.add(a.getID());
    }
    /**
     * Check whether there are duplicated default cases in one switch statement
     * @throws PGException
     */
    public void checkDefault() throws PGException{
        if(alreadyHasDefault){
            throw  new  PGException("There  is  already  a  default  case  in  the
current switch "
                                      +"statement");
        }
        returned = false;
        alreadyHasDefault = true;
    }

    /**
     * Enter a new function, and rebuild the return tree for checking whether
     * this function ever returns anything at the time of leaving function
     */
    public void enterFunction(){
        returnTree.clear();
    }

    /**
     * Leave the current function and check whether the function returns a data
     * type if it has one in any situations
     * @throws PGException
     */
    public void leaveFunction() throws PGException{
        if(currentFunction.equals(""))
            return;

        //returnTree.printTree();

        PGFunction func = (PGFunction) env.funcs.get(currentFunction);
        if(!func.getReturnType().toString().equals(VoidType) &&
           !returnTree.checkReturnExist(0)){
            String errMsg = "missing return statement for function '";
            StringTokenizer st = new StringTokenizer(currentFunction, "@");
            errMsg += st.nextToken() + "(";
            if(st.hasMoreTokens()){
                st = new StringTokenizer(st.nextToken(), "_");
                errMsg += st.nextToken();
                while(st.hasMoreTokens()){
                    errMsg += ", " + st.nextToken();
                }
            }
            errMsg += ")'";
            throw new PGException(errMsg);
        }
    }

    /**
     * Check whether a statement can be reached (If the statement comes after
     * return, break, or continue statements, then it can't be reached)
     */
    private void checkUnreacheableStatement(){
        if(returned){
            throw new PGException("Unreacheable statement");
        }
    }

    /**
```

```
     * Check whether a reserved word for Java is used for function, constant,
or
     * variable name
     * @param name String
     * @param title String
     */
    private void checkReservedWord(String name, String title){
        if(reservedWordList.contains(name)){
            throw new PGException("using '"+name+"' as a " + title + " name not
allowed");
        }
    }

    /**
     * End the switch statement
     */
    public void endSwitch(){
        currentCaseList.clear();
        if(!currentSwitchType.isEmpty())
            currentSwitchType.pop();
        if(!caseList.isEmpty())
            currentCaseList = (ArrayList)caseList.pop();
        alreadyHasDefault = false;
    }

    /**
     * Enter to a new local scope
     * @param func PGFunction
     * @param isLoop boolean
     * @param isSwitch boolean
     */
    public void enterScope(PGFunction func, boolean isLoop, boolean isSwitch){
        scopeLevel++;
        if(!isSwitch){
            returnTree.addNode();
            returnTree.enterScope(scopeLevel);
        }
        //System.out.println("EnterScope: Current scope level: " + scopeLevel);
        env.vars.enterScope();
        env.initVarEnterScope();
        env.arrayEnterScope();
        if(func != null){
            String  tempName  =  makeFuncName(func.getID(),  func.getArgTypes(),
true);
            currentFunction = tempName;
        }

        isLoopList.add(new Boolean(isLoop));
        isSwitchList.add(new Boolean(isSwitch));
    }

    /**
     * Leave the current scope
     */
    public void leaveScope(){
        //System.out.println("LeaveScope: Current scope level: " + scopeLevel);
        scopeLevel--;
        returned = false;
        env.arrayLeaveScope();
        env.initVarLeaveScope();
        env.vars.leaveScope();

        isLoopList.removeLast();
        isSwitchList.removeLast();
        returnTree.leaveScope(scopeLevel);
    }

    /**
     * Get the current enviroment variables
     * @return PGEnvironment
     */
    public PGEnvironment getEnv(){
        return env;
    }

    /**
     * Refreshes Environment variables, so it can be passed to analyze the next
     * pg file's semantics
     */
    public void refreshEnv(){
        env.funcLines.clear();
        env.globalLines.clear();
        env.pgTypeLines.clear();
        env.stringCaseLines.clear();
        env.switchLines.clear();
        env.charLines.clear();

        env.arrays.clear();
        env.defines_vals.clear();
        env.includes.clear();
        env.vars.getCurrentScope().clear();
        env.var_init_record.clear();

        returned = false;
    }

    /**
     * This is a helper function for debugging.
     */
    public void printEnv(){
        String id = "";
        // Print out includes
        Iterator i = env.includes.listIterator();
        System.out.println("/*******************                        Includes
*******************/");
        while(i.hasNext()){
            System.out.println(i.next());
        }

        // Print out defines
        i = env.defines.keySet().iterator();
        System.out.println("\n/**************         Defined         Constants
**************/");
        while(i.hasNext()){
            id = (String) i.next();
            System.out.println("ID:   "   +   id   +   "        DataType:   "   +
env.defines.get(id)
                                  + "   Value: " + env.defines_vals.get(id));
        }

        // Print out variables including globals and arrays
        i = env.vars.getCurrentScope().keySet().iterator();
        System.out.println("\n/***  Variables  (including  globals  and  arrays)
***/");
        while(i.hasNext()){
            id = (String) i.next();
            System.out.println("Variable ID: " + id + "   DataType: " +
                               env.vars.get(id) + "   Initialized: " +
                               env.var_init_record.get(id));
        }
        i = env.arrays.keySet().iterator();
        PGArray array = null;
        while(i.hasNext()){
            id = (String) i.next();
            array = (PGArray)env.arrays.get(id);
            System.out.println("Array ID: " + id + "   DataType: " +
                               array.getArrayType() + "   Dimension: " +
                               array.getDimensions() + "   Initialized: " +
                               env.var_init_record.get(id));
        }

        // Print out functions
        i = env.funcs.keySet().iterator();
        System.out.println("\n/*************** Functions ****************/");
        PGFunction function = null;
        Object [] arg_types;
        while(i.hasNext()){
            id = (String) i.next();
            function = (PGFunction)env.funcs.get(id);
            System.out.print("ID: " + function.getID() + "   ReturnType: " +
                             function.getReturnType() + "   ");
            arg_types = function.getArgTypes();
            if(arg_types != null){
                for (int j = 0; j < arg_types.length; j++) {
                    System.out.print("arg[" + j + "]: " + arg_types[j] + ", ");
                }
            }
            System.out.println();
        }
    }

    private String getSign(int sign){
        String s = "";
        switch(sign){
            case PLUS: s = "+"; break;
            case MULT: s = "*"; break;
            case MINUS: s = "-"; break;
            case DIV: s = "/"; break;
            case MOD: s = "%"; break;
            case GTE: s = ">="; break;
            case LTE: s = "<="; break;
            case GT: s = ">"; break;
            case LT: s = "<"; break;
            case EQ: s = "=="; break;
            case NEQ: s = "!="; break;
            case PPLUS: s = "++"; break;
            case MMINUS: s = "--"; break;
            case ASGN: s = "="; break;
            case MULTEQ: s = "*="; break;
            case PLUSEQ: s = "+="; break;
            case MINUSEQ: s = "-="; break;
            case DIVEQ: s = "/="; break;
            case MODEQ: s = "%="; break;
            case OREQ: s = "|="; break;
            case ANDEQ: s = "&="; break;
            case XOREQ: s = "^="; break;
            case AND: s = "&&"; break;
            case OR: s = "||"; break;
            case NOT: s = "!"; break;
            default:
        }

        return s;
    }
}
```

11.19 PGCodeGenerator

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: Convert .pg files to .java files</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.io.*;
import java.util.*;

public class PGCodeGenerator {

    private PGEnvironment env;
    private File pgFile;
    private File mainFile;

    private final String PhotogramMsg =
        "/*\n" +
        " * THIS FILE IS AUTOMATICALLY GENERATED BY PHOTOGRAM\n" +
        " */\n";

    public PGCodeGenerator(File mainFile, File pgFile, PGEnvironment env) {
        this.env = env;
        this.pgFile = pgFile;
        this.mainFile = mainFile;
    }

    public void generateJavaCode(){
        if(!pgFile.exists()){
            throw  new  PGException("file  "  +  pgFile.getName()  +  "  doesn't
exist.");
        }
        if(!mainFile.exists()){
            throw  new  PGException("file  "  +  mainFile.getName()  +  "  doesn't
exist.");
        }

        String className = changeToClassName(pgFile);

        try{
            File                 javaFile                 =                 new
File(mainFile.getAbsolutePath().replaceFirst(mainFile.getName(), "")
                                   + className + ".java");
            OutputStream os = new FileOutputStream(javaFile);

            String javaCodes = "";

            // First output the Photogram message to the java file
            os.write(PhotogramMsg.getBytes());
            os.flush();
            // Next output the imports (includes) if necessary (imports are
            //  required  only  if  the  included  files  are  in  the  different
directory
            // of this main pg file
            javaCodes = "import photogram.*;\n\n";
            os.write(javaCodes.getBytes());
            os.flush();

            // Now output the class name
            javaCodes = "public class " + className + " {\n";
            os.write(javaCodes.getBytes());
            os.flush();

            // Output defines (public static final)
            Iterator i = env.defines.keySet().iterator();
            String data_type = "";
            String id = "";
            String val = "";
            while(i.hasNext()){
                id = (String) i.next();
                // If the constant is defined in another file, then don't
                // create that constants in this file
                if(!pgFile.getName().equals((String)env.defines_class.get(id)))
                    continue;

                data_type = (String) env.defines.get(id);
                val = (String) env.defines_vals.get(id);
                if(!(data_type == SemanticAnalyzer.BooleanType ||
                     data_type == SemanticAnalyzer.IntegerType ||
                     data_type == SemanticAnalyzer.DoubleType ||
                     data_type == SemanticAnalyzer.StringType)){
                    data_type = "PG" + data_type;
                }
                if(data_type == SemanticAnalyzer.StringType)
                    val = "\"" + val + "\"";

                javaCodes = "\t" + "public static final " + data_type + " " +
                    id + " = " + val + ";\n";

                os.write(javaCodes.getBytes());
                os.flush();
            }

            // Output globals (either public or public static depends on
            // whether there is a "main" function in this pg file
            String access_specifier = "\t";
            boolean hasMain = env.funcs.containsKey("main@") |
                env.funcs.containsKey("main@_String[]");

            if(hasMain){
                PGFunction func = null;
                if(env.funcs.containsKey("main@"))
                    func = (PGFunction) env.funcs.get("main@");
                else if(env.funcs.containsKey("main@_String[]"))
                    func = (PGFunction) env.funcs.get("main@_String[]");

                if(!func.getClassName().equals(mainFile.getName()))
                    hasMain = false;
            }

            access_specifier = (hasMain) ? "public static " : "public ";
            // Now output the Photogram code in pg file
            // Create a BufferedReader from the fev file
            BufferedReader reader = new BufferedReader(new FileReader(pgFile));
            // Read first line
            javaCodes = reader.readLine();
            int line_num = 1;
            StringTokenizer st = null, st2 = null;
            String token = "", temp = "";
            boolean needCheck;
            while(javaCodes != null){
                // Ignore macros
                token = "";
                while(javaCodes.startsWith("#define")                         ||
javaCodes.startsWith("#include")){
                    javaCodes = reader.readLine();
                    line_num++;
                }

                needCheck = true;

                // If this line contains global value, then add appropriate
                // access specifier
                if(env.globalLines.contains(new Integer(line_num))){
                    //  Needs  to  take  care  of  front  tabs  '\t\  for  nice
formatting
                    javaCodes = access_specifier + " " +
                        javaCodes.replaceAll("\t", "");
                    needCheck = false;
                }

                // If this line contains function declaration, then add
                // access specifier 'public static'
                if(env.funcLines.contains(new Integer(line_num))){
                    boolean isMain = false;
                    if(hasMain){
                        temp = javaCodes;
                        st = new StringTokenizer(javaCodes, "\t (");
                        while(st.hasMoreTokens()){
                            if(st.nextToken().equals("main")){
                                isMain = true;
                                javaCodes  =  "public  static  void  main(String  []
args)";
                                if(temp.indexOf("{") != -1)
                                    javaCodes += "{";
                                break;
                            }
                        }
                    }

                    if(!hasMain || !isMain){
                        //  Needs  to  take  care  of  front  tabs  '\t'  for  nice
formatting
                        javaCodes = "public static " +
                            javaCodes.replaceAll("\t", "");
                    }

                    needCheck = false;
                }

                // If this line contains built-in object declaration or
                // built-in object function arguments, then convert to
                // 'PG'+type object type
                if(env.pgTypeLines.contains(new Integer(line_num))){
                    boolean doAgain = true;
                    while(doAgain){
                        doAgain = false;
                        st = new StringTokenizer(javaCodes, "\t ,(.[");
                        while (st.hasMoreTokens()) {
                            token = st.nextToken();
                            if (SemanticAnalyzer.isBuiltInObject(token)) {
                                if(javaCodes.indexOf(" "+token+" ") != -1){
                                    javaCodes  =  javaCodes.replaceAll("  "  +
token + " ", " PG" + token + " ");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("\t"+token+"  ")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("\t"  +
token + " ", "\tPG" + token + " ");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("  "+token+"(")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("  "  +
token + "\\(", " PG" + token + "(");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("  "+token+"[")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("  "  +
token + "\\[", " PG" + token + "[");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("  "+token+".")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("  "  +
token + "\\.", " PG" + token + ".");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("("+token+"  ")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("\\("  +
token + " ", "(PG" + token + " ");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf("("+token+".")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll("\\("  +
token + "\\.", "(PG" + token + ".");
                                    doAgain = true;
                                }
                                else  if(javaCodes.indexOf(","+token+".")  !=  -
1){
                                    javaCodes  =  javaCodes.replaceAll(","  +
token + "\\.", ",PG" + token + ".");
                                    doAgain = true;
                                }
                                else              if(env.funcLines.contains(new
Integer(line_num))){
                                    javaCodes = javaCodes.replaceAll(token+" ",
"PG" + token+" ");

                                    doAgain = true;
                                }
                                break;
                            }
                        }
                    }
                }

                // If this line contains switch statement that takes String
                // variable, then convert the variable to char
                if(env.switchLines.containsKey(new Integer(line_num))){
                    // This part is to make the .java file look better
                    // with correct front tab spaces
                    int tabCount = PGUtil.getFrontTabCount(javaCodes) - 1;
                    String frontTabs = "";
                    for(int j = 0; j < tabCount; j++)
                        frontTabs += "\t";
                    //      Add      a      new      char      type      named
"var_id"_Photogram_Generated_Char
                    // which takes the first char of the String "var_id"
                    // and add it to the new switch statement with this char
name
                    String     var_id     =     (String)env.switchLines.get(new
Integer(line_num));
                    //st = new StringTokenizer(javaCodes, "()");
                    String newSwitch = "";
                    //while(st.hasMoreTokens()){
                        //if(st.nextToken().equals("switch")){
                            newSwitch     =     "switch("     +     var_id     +
"_Photogram_Generated_Char){";
                        //}
                    //}
                    javaCodes   =   frontTabs   +   "char   "   +   var_id   +
"_Photogram_Generated_Char = "
                        +  var_id  +  ".charAt(0);\n"  +  frontTabs  +  "\t"  +
newSwitch;

                    needCheck = false;
                }

                // If this line contains String type case, then we need to
                // convert it to char case
                if(env.stringCaseLines.containsKey(new Integer(line_num))){
                    String   case_id   =   (String)env.stringCaseLines.get(new
Integer(line_num));
                    // Converts "case_id" to 'case_id' (i.e. "a" -> 'a')
                    javaCodes    =    javaCodes.replaceFirst("\""+case_id+"\"",
"'"+case_id+"'");
                }

                // Find PGLibrary functions and add "PGLibrary." infront of
                // these functions
                boolean doAgain = true;
                while (doAgain) {
                    doAgain = false;
                    st = new StringTokenizer(javaCodes, "\t ,(");
                    while (st.hasMoreTokens()) {
                        token = st.nextToken();
                        if (env.lib_func_list.contains(token)) {
                            if(javaCodes.indexOf(" "+token+"(") != -1){
                                javaCodes = javaCodes.replaceAll(token + "\\(",
"PGLibrary." + token + "(");
                                doAgain = true;
                            }
                            else if(javaCodes.indexOf("\t"+token+"(") != -1){
                                javaCodes  =  javaCodes.replaceAll("\t"+token  +
"\\(", "\tPGLibrary." + token + "(");
                                doAgain = true;
                            }
                            else if(javaCodes.indexOf("("+token+"(") != -1){
                                javaCodes = javaCodes.replaceAll("\\(" + token
+ "\\(", "(PGLibrary." + token + "(");
                                doAgain = true;
                            }
                            else if(javaCodes.indexOf(","+token+"(") != -1){
                                javaCodes = javaCodes.replaceAll("," + token +
"\\(", ",PGLibrary." + token + "(");
                                doAgain = true;
                            }
                            break;
                        }
                    }
                }

                // Convert String.charAt(int) object call to String since it's
originally char
                if(env.charLines.contains(new Integer(line_num))){
                    st = new StringTokenizer(javaCodes, "\t ,(");
                    while(st.hasMoreTokens()){
                        token = st.nextToken();
                        if(token.indexOf(".charAt") != -1){
                            st2 = new StringTokenizer(token, ".");
                            token = st2.nextToken();
                            st2 = new StringTokenizer(token, "[]");
                            token = st2.nextToken();
                            while(st2.hasMoreTokens()){
                                token += "\\[" + st2.nextToken() + "\\]";
                            }
                            javaCodes    =    javaCodes.replaceAll(token+"\\.",
"\"\"+" + token + ".");
                        }
                    }
                }

                // If this is not function declaration, global declaration, or
                // switch line, then check for functions or constants declared
                // in included files
                if(needCheck){
                    doAgain = true;
                    while (doAgain) {
                        doAgain = false;
                        st   =   new   StringTokenizer(javaCodes,   "\t   {}[]()+-
=/%&*^!|");
                        while (st.hasMoreTokens()) {
                            token = st.nextToken();
                            // Check constants declared outside of current pg
file
                            if(env.defines_class.containsKey(token)){
                                //  token+"("  check  is  used  to  distinguish
function
                                // name that is same as this constant name
                                if((javaCodes.indexOf(token+"(") == -1) &&

!pgFile.getName().equals((String)env.defines_class.get(token))){
                                    className                                 =
changeToClassName((String)env.defines_class.get(token));
                                    javaCodes   =   javaCodes.replaceAll(token,
className + "." + token);
                                    doAgain = true;
                                    break;
                                }
                            }
                            // Check functions declared outside of current pg
file
                            if(env.func_orig_list.containsKey(token)){
                                if((javaCodes.indexOf(token+"(") != -1) &&

!pgFile.getName().equals((String)env.func_orig_list.get(token))){
                                    className                              =
changeToClassName((String)env.func_orig_list.get(token));
                                    javaCodes                              =
javaCodes.replaceAll(token+"\\(", className+"."+token+"(");
                                    doAgain = true;
                                    break;
                                 }
                            }
                        }
                    }
                }

                javaCodes = "\t" + javaCodes + "\n";
                os.write(javaCodes.getBytes());
                os.flush();

                // Read next line
                javaCodes = reader.readLine();
                line_num++;
            }

            // Finally, close the class with end brace
            javaCodes = "}\n";
            os.write(javaCodes.getBytes());
            os.flush();

            // Close the output stream
            os.close();
        }
        catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    private String changeToClassName(String className){
        String name = PGUtil.removeExtension(className);
        String temps = className.toUpperCase();
        return name.replaceFirst(""+name.charAt(0), ""+temps.charAt(0));
    }

    private String changeToClassName(File file){
        String className = PGUtil.removeExtension(file);
        // Make the first letter of the class name Capital
        String temps = className.toUpperCase();
        return                    className.replaceFirst(""+className.charAt(0),
""+temps.charAt(0));
    }
}
```

11.20 PGCompiler

```
package photogram;

//Think about drawImage
```

```
import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class PGCompiler {
    public static void main(String args[]) {
        try {
            File file = new File(args[0]);
            BufferedReader pgFile = new BufferedReader(new FileReader(file));

            // Create the lexer and parser and feed them the input
            PGAntlrLexer lexer = new PGAntlrLexer(pgFile);
            PGAntlrParser parser = new PGAntlrParser(lexer);
            parser.setASTNodeClass(CommonASTWithLines.class.getName());
            parser.program(); // "file" is the main rule in the parser

            if(!parser.errMsg.equals("")){
                System.out.println(parser.errMsg);
                return;
            }

            // Get the AST from the parser
            CommonASTWithLines       parseTree       =       (CommonASTWithLines)
parser.getAST();

            // Print the AST in a human-readable format
            if(parseTree == null)
                return;

            SemanticAnalyzer sa = new SemanticAnalyzer();

            PrePGAntlrWalker prewalker = null;
            PGAntlrWalker walker = null;
            try {
                prewalker = new PrePGAntlrWalker();
                prewalker.sa = sa;
                prewalker.setASTNodeClass(CommonASTWithLines.class.getName());
                prewalker.mainFile = file;
                prewalker.currFile = file;
                prewalker.expr(parseTree);
                System.out.println(prewalker.errMsg);

                if(!prewalker.errMsg.equals(""))
                    return;

                walker = new PGAntlrWalker();
                walker.sa = sa;
                walker.setASTNodeClass(CommonASTWithLines.class.getName());
                walker.expr(parseTree);
                System.out.println(walker.errMsg);
            }
            catch (PGException e) {
                //e.printStackTrace();
            }
            catch (Exception e) {
                //e.printStackTrace();
            }

            if(prewalker.errMsg.equals("") && walker.errMsg.equals("")){
                PGCodeGenerator codeGenerator = new PGCodeGenerator(file, file,
sa.getEnv());
```

```
                codeGenerator.generateJavaCode();
            }
        }
        catch (Exception e) {
            //System.err.println("Exception: " + e);
            //e.printStackTrace();
        }
    }
}
```

11.21 ColorFreq

```
/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
package photogram;

public class ColorFreq implements Comparable {

    public int id;
    public int freq;

    public ColorFreq(int id, int freq) {
        this.id = id;
        this.freq = freq;
    }

    public int compareTo(Object o) {
        ColorFreq cf = (ColorFreq) o;
        if (this.freq < cf.freq)
            return -1;
        else if (this.freq == cf.freq)
            return 0;
        else
            return 1;
    }
}
```

11.22 CommonASTWithLines

```
package photogram;

import antlr.CommonAST;
import antlr.Token;

public class CommonASTWithLines extends CommonAST {
     private int line = 0;
     private int column = 0 ;
     public void initialize(Token tok) {
           super.initialize(tok);
           line=tok.getLine();
          column = tok.getColumn();
      }
     public int getLine(){
        return line;
     }
     public int getColumn() {
        return column;
     }
}
```

11.23 ImageFileFilter

```
package photogram;

import java.io.File;

public class ImageFileFilter implements java.io.FileFilter{
    public boolean accept(File f) {
        if (f.isDirectory()) {
            return true;
        }

        String extension = PGUtil.getExtension(f);
        if (extension != null) {
            if (extension.equals("jpg") ||
                extension.equals("jpeg") ||
                extension.equals("gif") ||
                extension.equals("png")){
                return true;
            }
            else {
                return false;
            }
        }
        return false;
    }

    public String getDescription() {
        return "Image Files (*.jpg, *.jpeg, *.gif, *.png)";
    }

}
```

11.24 PGLibrary

```
package photogram;

/**
 * <p>Title: </p>
 * <p>Description: A collection of built-in functions useful for developing
 * Photogram functions.</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
import java.io.*;
import java.util.*;
import java.awt.*;

public class PGLibrary {

    /*********************************************************************
     *      Useful Functionalities for Developping Photogram Functions    *
     *********************************************************************/

    /******************* BASIC MATHEMATICAL FUNCTIONS *******************/
```

```
/**
 * Returns a random double number in the range of 0 to 1
 * @return double
 */
public static double random() {
    return Math.random();
}

/**
 * Returns cos(radian)
 * @param radian double
 * @return double
 */
public static double cos(double radian) {
    return Math.cos(radian);
}

/**
 * Return sin(radian)
 * @param radian double
 * @return double
 */
public static double sin(double radian) {
    return Math.sin(radian);
}

public static double tan(double radian) {
    return Math.tan(radian);
}

public static double acos(double val) {
    return Math.acos(val);
}

public static double asin(double val) {
    return Math.asin(val);
}

public static double ceil(double val) {
    return Math.ceil(val);
}

public static double floor(double val) {
    return Math.floor(val);
}

public static double sqrt(double val) {
    return Math.sqrt(val);
}

public static double abs(double val) {
    return Math.abs(val);
}

public static double pow(double val, double power) {
    return Math.pow(val, power);
}

public static int pow(int val, int power) {
    return (int)Math.pow((double)val, (double)power);
}
```

```
    public static double log(double val) {
        return Math.log(val);
    }

    public static double exp(double val) {
        return Math.exp(val);
    }

    public static double pi() {
        return Math.PI;
    }

    public static double e() {
        return Math.E;
    }

    /***************** END OF BASIC MATHEMATICAL FUNCTIONS ******************/

    public static PGImage[] openDir(String dirName) {
        PGImage[] images = null;
        File dir = new File(dirName);
        if (!dir.exists()) {
            throw new PGException("Directory " + dirName + " doesn't exist.");
        }
        if (!dir.isDirectory()) {
            throw new PGException(dirName + " is not a directory.");
        }

        // Get all of the image files from this directory
        FileFilter filter = new ImageFileFilter();
        File[] imgFiles = dir.listFiles(filter);
        if (imgFiles != null) {
            images = new PGImage[imgFiles.length];
            for (int i = 0; i < images.length; i++) {
                images[i] = new PGImage();
                images[i].open(imgFiles[i].getAbsolutePath(),
PGImage.RGB_TYPE);
            }
        }

        return images;
    }

    public static int doubleToInteger(double a) {
        return (int) a;
    }

    public static int maskBits(int bits, int num_mask_bits) {
        return (bits << (32 - num_mask_bits)) >> (32 - num_mask_bits);
    }

    public static PGImage drawShape(final PGImage src, PGLine line,
                                    PGColor color) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        if (line == null) {
            throw new PGException("line is null");
        }

        if (color == null) {
            color = new PGColor(0, 0, 0);
        }
        PGImage dest = new PGImage();
        dest.copy(src);

        Graphics2D g2d = dest.getBufferedImage().createGraphics();
        g2d.setColor(new Color(color.r, color.g, color.b, color.a));
        g2d.drawLine(line.sx, line.sy, line.ex, line.ey);

        return dest;
    }

    public static PGImage drawShape(final PGImage src, PGOval oval,
                                    PGColor color,
                                    boolean fill) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        if (oval == null) {
            throw new PGException("oval is null");
        }

        if (color == null) {
            color = new PGColor(0, 0, 0);
        }

        PGImage dest = new PGImage();
        dest.copy(src);

        Graphics2D g2d = dest.getBufferedImage().createGraphics();
        g2d.setColor(new Color(color.r, color.g, color.b, color.a));
        if (fill) {
            g2d.fillOval(oval.x, oval.y, oval.a, oval.b);
        } else {
            g2d.drawOval(oval.x, oval.y, oval.a, oval.b);
        }

        return dest;
    }

    public static PGImage drawShape(final PGImage src, PGRect rect,
                                    PGColor color,
                                    boolean fill) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        if (rect == null) {
            throw new PGException("rectangle is null");
        }

        if (color == null) {
            color = new PGColor(0, 0, 0);
        }

        PGImage dest = new PGImage();
        dest.copy(src);

        Graphics2D g2d = dest.getBufferedImage().createGraphics();
        g2d.setColor(new Color(color.r, color.g, color.b, color.a));
        if (fill) {
            g2d.fillRect(rect.x, rect.y, rect.width, rect.height);
        } else {
            g2d.drawRect(rect.x, rect.y, rect.width, rect.height);
        }
        return dest;
    }

    /**
     * Draw a text onto the source image starting from coordinate (x, y) of
     * the image with a specific font and color
     * @param src PGImage the source image
     * @param text PGString the text
     * @param x int the starting x-coordinate
     * @param y int the starting y-coordinate
     * @param font PGFont the font of the text
     * @param color PGColor the color of the text
     * @return PGImage
     */
    public static PGImage drawText(final PGImage src, String text, int x, int
y,
                                PGFont font, PGColor color) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        if (text == null) {
            throw new PGException("text string is null");
        }

        if (color == null) {
            color = new PGColor(0, 0, 0);
        }

        if (font == null) {
            font = new PGFont();
        }

        PGImage dest = new PGImage();
        dest.copy(src);

        Graphics2D g2d = dest.getBufferedImage().createGraphics();
        g2d.setColor(new Color(color.r, color.g, color.b, color.a));
        g2d.setFont(new Font(font.family, font.type, font.size));
        g2d.drawString(text.toString(), x, y);

        return dest;
    }

    /**
     * Resize the source image to a specific width and height (aspect ratio
     * is not reserved)
     * @param src PGImage the source image
     * @param width int the resized width
     * @param height int the resized height
     * @return PGImage
     */
    public static PGImage resize(final PGImage src, int width, int height) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        PGImage dest = new PGImage();

        return dest;
    }
    public static PGImage paste(PGImage dest, final PGImage src, int x_pos,
                                int y_pos) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        if (dest == null) {
            throw new PGException("destination image is null");
        }

        if (x_pos < 0 || x_pos >= dest.getWidth() || y_pos < 0 ||
            y_pos >= dest.getHeight()) {
            throw new PGException("x_pos or y_pos is out of boundary: x_pos = "
                                  + x_pos + ", y_pos = " + y_pos);
        }
        PGRect    rect    =    new    PGRect(x_pos,    y_pos,    src.getWidth(),
src.getHeight());
        return paste(dest, src, rect);
    }

    public static PGImage paste(PGImage dest, final PGImage src, PGRect rect) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        if (dest == null) {
            throw new PGException("destination image is null");
        }
        if (rect == null) {
            throw new PGException("rectangle is null");
        }
        if (rect.x < 0 || rect.x >= dest.getWidth() || rect.y < 0 ||
            rect.y >= dest.getHeight()) {
            throw new PGException(
                    "rect.x or rect.y is out of boundary: rect.x = "
                    + rect.x + ", rect.y = " + rect.y);
        }

        PGImage new_img = new PGImage();
        new_img.copy(dest);

        int dest_width = dest.getWidth();
        int src_width = src.getWidth();
        int dest_height = dest.getHeight();
        int src_height = src.getHeight();

        int idx_dest, idx_src;
        for (int i = rect.y; (i < (rect.y + rect.height)) && (i < src_height)
                     && (i < dest_height); i++) {
            for (int j = rect.x; (j < (rect.x + rect.width)) && (j < src_width)
                         && (j < dest_width); j++) {
                idx_src = i * src_width + j;
                idx_dest = i * dest_width + j;
                new_img.getData()[idx_dest] = src.getData()[idx_src];
            }
        }

        return new_img;
    }

    /**
     * Rotate the source image clockwise for a specified degree
     * @param src PGImage
     * @param degrees double
     * @return PGImage
```

```
 */
public static PGImage rotate(PGImage src, double degrees) {
    if (src == null) {
        throw new PGException("source image is null");
    }

    PGImage dest = new PGImage(src.getWidth(), src.getHeight());

    degrees %= 360;

    int[] srcData = src.getData();
    int[] destData = dest.getData();
    int width = src.getWidth();
    int height = src.getHeight();

    int x, y;
    double cos, sin;
    for (int i = -height / 2; i < height / 2; i++) {
        for (int j = -width / 2; j < width / 2; j++) {
            if (degrees == 0) {
                cos = 1;
                sin = 0;
            } else if (degrees == 90) {
                cos = 0;
                sin = 1;
            } else if (degrees == 180) {
                cos = -1;
                sin = 0;
            } else if (degrees == 270) {
                cos = 0;
                sin = -1;
            } else {
                cos = Math.cos(degrees * Math.PI / 180);
                sin = Math.sin(degrees * Math.PI / 180);
            }

            x = (int) (cos * j - sin * i);
            y = (int) (sin * j + cos * i);
            y += height / 2;
            x += width / 2;
            if (x >= 0 && x < width && y >= 0 && y < height) {
                destData[y * width +
                        x] = srcData[(i + height / 2) * width +
                             (j + width / 2)];
            }
        }
    }

    return filterGaussian(dest);
}

/**
 * Modify the color of the image using the given colorMap
 * @param src PGImage
 * @param colorMap int[]
 * @return PGImage
 */
public static PGImage map(final PGImage src, int[] colorMap) {
    if (src == null) {
        throw new PGException("source image is null");
    }

    PGImage dest = new PGImage(src.getWidth(), src.getHeight());
```

```
        int[] srcData = src.getData();
        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        double closest = 0;
        int match = 0;
        double distance;
        int r, g, b, rc, gc, bc;
        Map closestMap = new HashMap();
        // Compute the closest color from the color map for each pixel and
assign
        // the colors to the pixels
        for (int i = 0; i < height * width; i++) {
            // Check whether the closest color for this src color has
            // already been calculated, if so, then just use from the
            // closestMap for speed optimization
            if (closestMap.containsKey(new Integer(srcData[i]))) {
                destData[i] = Integer.parseInt("" + closestMap.get(
                        new Integer(srcData[i])));
            } else {
                r = (srcData[i] >> 16) & 0x00FF;
                g = (srcData[i] >> 8) & 0x00FF;
                b = (srcData[i] >> 0) & 0x00FF;
                // Find the closed color from the color map
                for (int k = 0; k < colorMap.length; k++) {
                    rc = (colorMap[k] >> 16) & 0x00FF;
                    gc = (colorMap[k] >> 8) & 0x00FF;
                    bc = (colorMap[k] >> 0) & 0x00FF;
                    distance = Math.sqrt(Math.pow((r - rc), 2) +
                                         Math.pow((g - gc), 2)
                                         + Math.pow((b - bc), 2));
                    if (k == 0) {
                        closest = distance;
                        match = 0;
                    } else {
                        if (distance < closest) {
                            closest = distance;
                            match = k;
                        }
                    }
                }

                // Assign the closest color
                destData[i] = colorMap[match];
                closestMap.put(new Integer(srcData[i]),
                               new Integer(colorMap[match]));
            }
        }

        return dest;
    }

    /**
     * Convolute the given image with a given mask
     * @param src PGImage
     * @param mask double[][]
     * @return PGImage
     */
    public static PGImage convol(final PGImage src, double[][] mask) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // If the mask is null or has zero row or column, return original image
        if (mask == null || mask.length < 1 || mask[0].length < 1) {
            throw new PGException(
                    "mask  is  null  or  one  or  two  of  its  dimension(s)  is(are)
less than 0");
        }

        // If the mask length has even numbers, then return original image
        /*if((mask.length % 2) == 0 || (mask[0].length % 2) == 0)
            return src;*/

        PGImage dest = new PGImage(src.getWidth(), src.getHeight());

        int[] srcData = src.getData();
        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        int idx, idx2, tmp;
        double sum = 0;
        int ibeg, iend, jbeg, jend;
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                idx = y * width + x;
                // if it's around the edge, it will use a smaller box
                if (x < (mask[0].length / 2)) {
                    jbeg = -x;
                } else {
                    jbeg = -(mask[0].length / 2);
                }

                if (x >= (width - mask[0].length / 2)) {
                    jend = ((width - 1) - x);
                } else {
                    jend = mask[0].length / 2;
                }

                if (mask[0].length % 2 == 0) {
                    jend--;
                }

                if (y < (mask.length / 2)) {
                    ibeg = -y;
                } else {
                    ibeg = -(mask.length / 2);
                }

                if (y >= (height - mask.length / 2)) {
                    iend = (height - 1) - y;
                } else {
                    iend = mask.length / 2;
                }

                if (mask.length % 2 == 0) {
                    iend--;
                }

                for (int color = 0; color <= 16; color += 8) {
                    sum = 0;
                    for (int i = ibeg; i <= iend; i++) {
                        for (int j = jbeg; j <= jend; j++) {
```

```
                        idx2 = idx + i * width + j;
                        tmp = (srcData[idx2] >> color) & 0x00FF;
                        sum += tmp * mask[i + mask.length / 2][j +
                                mask[0].length / 2];
                    }
                }
                if (sum > 255) {
                    sum = 255;
                } else if (sum < 0) {
                    sum = 0;
                }
                destData[idx] |= ((int) sum << color);
            }

            // Preserve the alpha value
            destData[idx] |= ((srcData[idx] >> 24) << 24);
        }
    }

    return dest;
}

public static void print(String msg) {
    System.out.print(msg);
}

public static void println(String msg) {
    System.out.println(msg);
}

/*********************************************************************
 *               Built-in Image Processing Functionalities              *
 *********************************************************************/

/**
 * Convert the source image to a grayscaled image
 * @param src PGImage
 * @return PGImage
 */
public static PGImage grayscale(final PGImage src) {
    if (src == null) {
        throw new PGException("source image is null");
    }

    PGImage dest = new PGImage(src.getWidth(), src.getHeight());

    int[] destData = dest.getData();
    int[] srcData = src.getData();
    int r, g, b, gray;
    int width = src.getWidth();
    int height = src.getHeight();
    for (int i = 0; i < height * width; i++) {
        r = (srcData[i] >> 16) & 0x00FF; // Red Component
        g = (srcData[i] >> 8) & 0x00FF; // Green Component
        b = (srcData[i] >> 0) & 0x00FF; // Blue Component
        // R, G, and B are all equal to the intensity for the grayscale
        gray = (int) (0.299 * r + 0.587 * g + 0.114 * b);
        destData[i] = (gray << 16) | (gray << 8) | gray;
    }

    return dest;
}
```

```
    /**
     * Quantize the source image into 256 color image using an uniformally
distributed
     * color map
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage quantUniform(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        PGImage dest = new PGImage(src.getWidth(), src.getHeight());

        int[] srcData = src.getData();
        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();
        int r, g, b;
        for (int i = 0; i < height * width; i++) {
            r = ((srcData[i] >> 16) & 0x00FF) / 32; // Get 8 shades (3 bits) of
Red Component
            g = ((srcData[i] >> 8) & 0x00FF) / 32; // Get 8 shades (3 bits) of
Green Component
            b = ((srcData[i] >> 0) & 0x00FF) / 64; // Get 4 shades (2 bits) of
Blue Component
            destData[i] = ((r * 32) << 16) | ((g * 32) << 8) | (b * 64);
        }

        return dest;
    }

    /**
     * Quantize the source image into 256 color image using the 256 most
popular
     * color used in this image
     * (NOTE: NOT WORKING YET)
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage quantPopulosity(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        int[] srcData = src.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        Map histogram = new HashMap();
        Map keyMap = new HashMap();

        ColorFreq value = null;
        int r, g, b, color;
        // Quantize uniformally and build the color histogram
        for (int i = 0; i < height * width; i++) {
            r = ((srcData[i] >> 16) & 0x00FF) >> 3; // Get 32 shades (5 bits)
of Red Component
            g = ((srcData[i] >> 8) & 0x00FF) >> 3; // Get 32 shades (5 bits) of
Green Component
            b = ((srcData[i] >> 0) & 0x00FF) >> 3; // Get 32 shades (5 bits) of
Blue Component
            color = (r << 16) | (g << 8) | (b);
            if (histogram.containsKey(new Integer(color))) {
                value = (ColorFreq) histogram.get(new Integer(color));
                // Increment the frequency of this color
                value.freq++;
            } else {
                value = new ColorFreq(histogram.size(), 0);
                keyMap.put(new Integer(value.id), new Integer(color));
                histogram.put(new Integer(color), value);
            }
        }

        Object[] sortedHistogram = histogram.values().toArray();

        Arrays.sort(sortedHistogram, new Comparator() {
            public int compare(Object o1, Object o2) {
                return ((ColorFreq) o2).freq - ((ColorFreq) o1).freq;
            }
        });

        int[] colorMap = new int[256];
        ColorFreq highest = null;
        // Get the 256 most popular colors and create a color with 256 values
        for (int i = 0; i < 256 && i < sortedHistogram.length; i++) {
            highest = (ColorFreq) sortedHistogram[i];
            color = Integer.parseInt("" + keyMap.get(new Integer(highest.id)));
            r = ((color >> 16) & 0x00FF) << 3;
            g = ((color >> 8) & 0x00FF) << 3;
            b = ((color >> 0) & 0x00FF) << 3;
            colorMap[i] = (r << 16) | (g << 8) | (b);
        }

        // If the image has less than 256 colors, fill the remaing color map
        // table with black color
        for (int i = sortedHistogram.length; i < 256; i++) {
            colorMap[i] = 0;
        }

        return map(src, colorMap);
    }

    /**
     * Dither the source image using threshold of 0.5
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage ditherThreshold(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Get the grayscaled image
        PGImage dest = grayscale(src);

        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        double intensity;
        for (int i = 0; i < height * width; i++) {
            intensity = (destData[i] & 0x00FF) / 255.0;
            if (intensity < 0.5) {
                destData[i] = 0; // convert to Black
            } else {
```

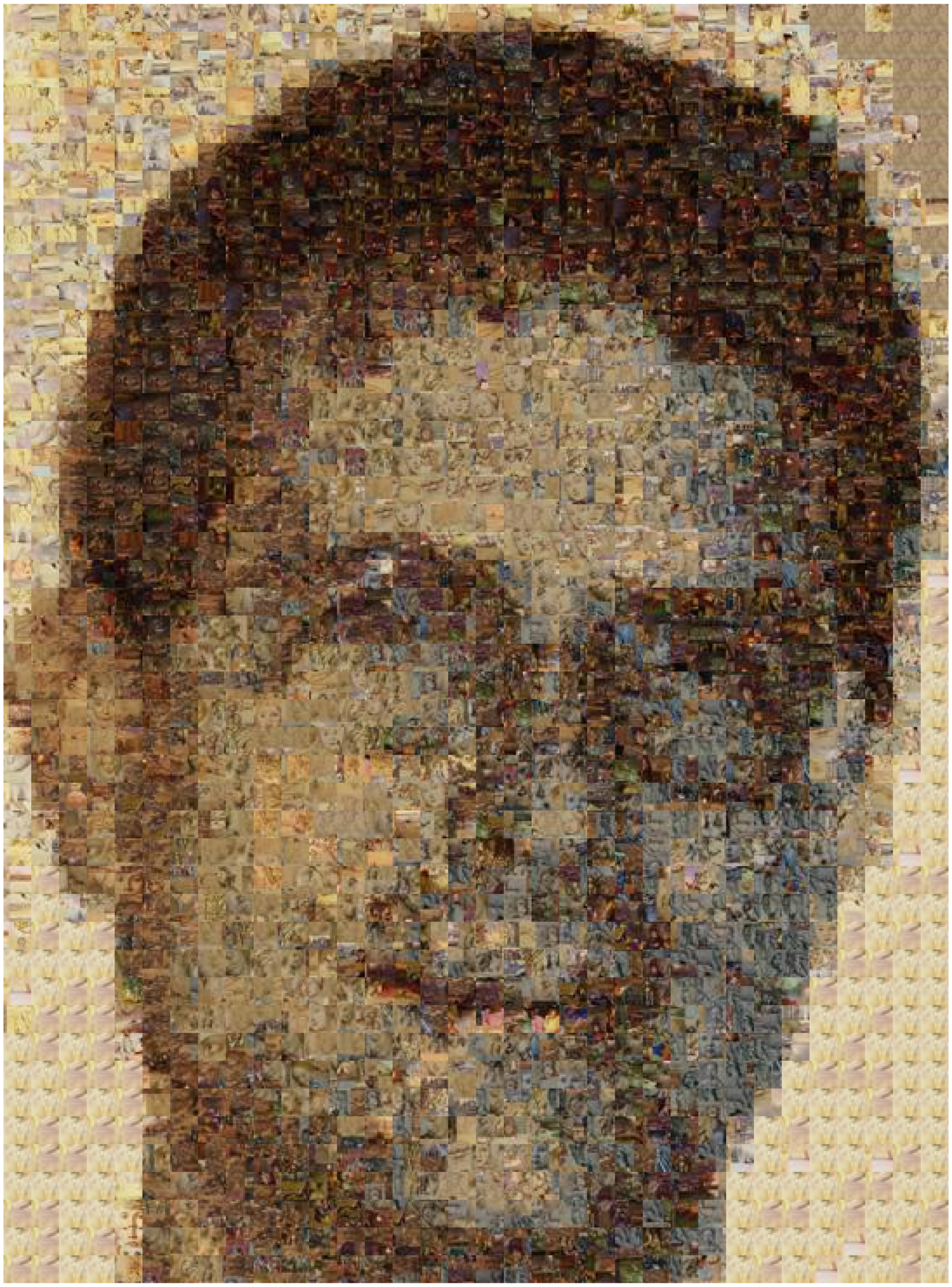
```
            destData[i] = 0x00FFFFFF; // convert to White
        }
    }

    return dest;
}

/**
 * Dither the source image randomly
 * @param src PGImage
 * @return PGImage
 */
public static PGImage ditherRandom(final PGImage src) {
    if (src == null) {
        throw new PGException("source image is null");
    }

    // Get the grayscaled image
    PGImage dest = grayscale(src);

    int[] destData = dest.getData();
    int width = src.getWidth();
    int height = src.getHeight();

    double intensity;
    double random_val;

    for (int i = 0; i < height * width; i++) {
        // Get a random value that ranges from -0.2 to 0.2
        random_val = ((double) random() * 0.4) - 0.2;
        // Add random values to each pixel uniformaly
        intensity = ((destData[i] & 0x00FF) + random_val * 255) / 255.0;
        if (intensity < 0.5) {
            destData[i] = 0; // convert to Black
        } else {
            destData[i] = 0x00FFFFFF; // convert to White
        }
    }

    return dest;
}

/**
 * Dither the source image using Floyd-Steinberg technique
 * @param src PGImage
 * @return PGImage
 */
public static PGImage ditherFS(final PGImage src) {
    if (src == null) {
        throw new PGException("source image is null");
    }

    // Get the grayscaled image
    PGImage dest = grayscale(src);

    int[] destData = dest.getData();
    int width = src.getWidth();
    int height = src.getHeight();

    double intensity = 0.0;
    int idx;
    double error = 0.0;
    double tmp;
```

```
        for (int i = 0; i < height; i++) {
            // Do zigzag ordering
            if ((i % 2) == 0) {
                // from left to right
                for (int j = 0; j < width; j++) {
                    idx = i * width + j;
                    intensity = (destData[idx] & 0x00FF) / 255.0;

                    if (intensity < 0.5) {
                        destData[idx] = 0; // convert to Black
                    } else {
                        destData[idx] = 0x00FFFFFF; // convert to White
                    }

                    error = intensity - (destData[idx] & 0x00FF) / 255.0;

                    // if it's not at the right edge, then propagate the 7/16
error to the right pixel
                    if (j != (width - 1)) {
                        tmp = ((destData[idx + 1] & 0x00FF) / 255.0) +
                              (7 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + 1] = (int) (tmp * 255);
                    }
                    // if it's neither at the right or bottom edge of the
image, propagate 1/16 error
                    // to the bottom-right pixel
                    if ((j != (width - 1)) && (i != (height - 1))) {
                        tmp = ((destData[idx + width + 1] & 0x00FF) / 255.0) +
                              (1 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width + 1] = (int) (tmp * 255);
                    }
                    // if it's not at the bottom edge of the image, then
propagate 5/16 error to the
                    // bottom pixel
                    if (i != (height - 1)) {
                        tmp = ((destData[idx + width] & 0x00FF) / 255.0) +
                              (5 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width] = (int) (tmp * 255);
                    }
                    // if it's neither at the left edge or bottom edge of the
image, then propagate
                    // 3/16 error to the bottom-left pixel
                    if ((j != 0) && (i != (height - 1))) {
                        tmp = ((destData[idx + width - 1] & 0x00FF) / 255.0) +
                              (3 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width - 1] = (int) (tmp * 255);
                    }
                }
            } else {
                // from right to left
                for (int j = (width - 1); j >= 0; j--) {
                    idx = i * width + j;
                    intensity = (destData[idx] & 0x00FF) / 255.0;

                    if (intensity < 0.5) {
                        destData[idx] = 0; // convert to Black
                    } else {
                        destData[idx] = 0x00FFFFFF; // convert to White
                    }

                    error = intensity - (destData[idx] & 0x00FF) / 255.0;

                    // if it's not at the left edge, then propagate the 7/16
error to the left pixel
                    if (j != 0) {
                        tmp = ((destData[idx - 1] & 0x00FF) / 255.0) +
                              (7 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx - 1] = (int) (tmp * 255);
                    }
                    // if it's neither at the left or bottom edge of the image,
propagate 1/16 error
                    // to the bottom-left pixel
                    if ((j != 0) && (i != (height - 1))) {
                        tmp = ((destData[idx + width - 1] & 0x00FF) / 255.0) +
                              (1 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width - 1] = (int) (tmp * 255);
                    }
                    // if it's not at the bottom edge of the image, then
propagate 5/16 error to the
                    // bottom pixel
                    if (i != (height - 1)) {
                        tmp = ((destData[idx + width] & 0x00FF) / 255.0) +
                              (5 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width] = (int) (tmp * 255);
                    }
                    // if it's neither at the right edge or bottom edge of the
image, then propagate
                    // 3/16 error to the bottom-right pixel
                    if ((j != (width - 1)) && (i != (height - 1))) {
                        tmp = ((destData[idx + width + 1] & 0x00FF) / 255.0) +
                              (3 * error / 16);
                        if (tmp > 1.0) {
                            tmp = 1.0;
                        } else if (tmp < 0.0) {
                            tmp = 0.0;
                        }

                        destData[idx + width + 1] = (int) (tmp * 255);
                    }
                }
            }
        }

        return dest;
    }

    /**
     * Dither the source image while conserving the brightness value of the
     * source image
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage ditherBright(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Get the grayscaled image
        PGImage dest = grayscale(src);

        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        double sum = 0;
        int[] gray = new int[256];
        // Initialize all of the elements to 0
        for (int i = 0; i < 256; i++) {
            gray[i] = 0;
        }

        // Get the average intensity of the original image
        for (int i = 0; i < height * width; i++) {
            gray[destData[i] & 0x00FF]++;
            sum += destData[i] & 0x00FF;
        }

        double brightness = (sum / (height * width)) / 255;
        int s = 0;
        int k;
        int num_white = (int) (sum / 255);
        for (k = 255; k >= 0 && s < num_white; k--) {
            s += gray[k];
        }

        double threshold = (k + 1) / 255.0;
        double intensity;
        double tmp = 0;
        // Dither the image using the average threshold
        for (int i = 0; i < height * width; i++) {
            intensity = (destData[i] & 0x00FF) / 255.0;
            if (intensity < threshold) {
                destData[i] = 0; // convert to Black
            } else {
                destData[i] = 0x00FFFFFF; // convert to White
            }

            tmp += destData[i] & 0x00FF;
        }

        return dest;
    }

    public static PGImage ditherBlock(final PGImage src, double[][] mask) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // If the mask is null or has zero row or column, return black image
        if (mask == null || mask.length < 1 || mask[0].length < 1) {
            return new PGImage(src.getWidth(), src.getHeight());
        }
        // Get the grayscaled image
        PGImage dest = grayscale(src);

        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        double intensity;

        int idx;
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                idx = i * width + j;
                intensity = (destData[idx] & 0x00FF) / 255.0;
                if (intensity < mask[j % mask.length][i % mask[0].length]) {
                    destData[idx] = 0; // convert to Black
                } else {
                    destData[idx] = 0x00FFFFFF; // convert to White
                }
            }
        }

        return dest;
    }

    /**
     * Dither the source image using a 4x4 ordered mask matrix
     * @param src PGImage
     * @return PGImag
     */
    public static PGImage ditherOrdered(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        double[][] mask = { {0.1250, 1.0000, 0.1875, 0.8125}, {0.6250, 0.3750,
                            0.6875, 0.4375}, {0.2500, 0.8750, 0.0625, 0.9375},
                            {0.7500, 0.5000, 0.5625, 0.3125}
        };

        return ditherBlock(src, mask);
    }

    public static PGImage ditherCluster(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        double[][] mask = { {0.7500, 0.3750, 0.6250, 0.2500}, {0.0625, 1.0000,
                            0.8750, 0.4375}, {0.5000, 0.8125, 0.9375, 0.1250},
                            {0.1875, 0.5625, 0.3125, 0.6875}
        };

        return ditherBlock(src, mask);
    }

    public static PGImage filterBox(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Creates a mask for the filter
        double[][] mask = new double[5][5];
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                mask[i][j] = 1.0 / 25.0;
            }
        }

        return convol(src, mask);
    }

    public static PGImage filterBartlett(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Creates a mask for the filter
        double[][] mask = { {1, 2, 3, 2, 1}, {2, 4, 6, 4, 2}, {3, 6, 9, 6, 3},
                            {2, 4, 6, 4, 2}, {1, 2, 3, 2, 1}
        };

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                mask[i][j] /= 81.0;
            }
        }

        return convol(src, mask);
    }

    public static PGImage filterGaussian(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Creates a mask for the filter
        double[][] mask = { {1, 4, 6, 2, 1}, {4, 16, 24, 16, 4}, {6, 24, 36,
24,
                            6}, {4, 16, 24, 16, 4}, {1, 4, 6, 2, 1}
        };
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                mask[i][j] /= 256.0;
            }
        }

        return convol(src, mask);
    }

    public static PGImage filterGaussianN(final PGImage src, int n) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Creates a mask for the filter
        double[][] mask = new double[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mask[i][j] = PGUtil.Binomial((n - 1), i) *
                             PGUtil.Binomial((n - 1), j) /
                             Math.pow(Math.pow(2, (n - 1)), 2);
            }
        }

        return convol(src, mask);
    }

    /**
     * Detect the edge of the source image
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage filterEdgeDetect(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Creates a mask for the filter
        double[][] mask = { { -1, -4, -6, -2, -1}, { -4, -16, -24, -16, -4},
                          { -6, -24, 220, -24, -6}, { -4, -16, -24, -16, -4},
                          { -1, -4, -6, -2, -1}
        };

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                mask[i][j] /= 256.0;
            }
        }

        return convol(src, mask);
    }

    /**
     * Enhance the edge of the source image.
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage filterEnhance(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }
        // Creates a mask for the filter
        double[][] mask = { { -1, -4, -6, -2, -1}, { -4, -16, -24, -16, -4},
                          { -6, -24, 476, -24, -6}, { -4, -16, -24, -16, -4},
                          { -1, -4, -6, -2, -1}
        };

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                mask[i][j] /= 256.0;
            }
        }

        return convol(src, mask);
    }

    /**
     * Half the size of the source image
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage sizeHalf(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        // Get the bartlett filtered image
        PGImage tmp = filterBartlett(src);

        int[] tmpData = tmp.getData();
        int width = (src.getWidth() >> 1) << 1; // make is multiple of 2
        int height = (src.getHeight() >> 1) << 1; // make is multiple of 2

        int destWidth, destHeight;
        if (src.getWidth() % 2 == 1) {
            destWidth = width / 2 + 1;
        } else {
            destWidth = width / 2;
        }
        destHeight = height / 2;

        PGImage dest = new PGImage(destWidth, destHeight);
        int[] destData = dest.getData();

        int idx, idx2;
        for (int i = 0; i < height; i += 2) {
            for (int j = 0; j < width; j += 2) {
                idx = i * width + j;
                idx2 = (i * width) / 4 + j / 2;
                destData[idx2] = tmpData[idx];
            }
        }

        return dest;
    }

    /**
     * Double the size of the source image
     * @param src PGImage
     * @return PGImage
     */
    public static PGImage sizeDouble(final PGImage src) {
        if (src == null) {
            throw new PGException("source image is null");
        }

        PGImage dest = new PGImage(src.getWidth() * 2, src.getHeight() * 2);

        int[] srcData = src.getData();
        int[] destData = dest.getData();
        int width = src.getWidth();
        int height = src.getHeight();

        int couI = 0, couJ = 0;
        int idx;
        for (int i = 0; i < height; i++) {
            couI = 0;
            for (int j = 0; j < width; j++) {
                idx = i * width + j;
                destData[(couJ * (width * 2) + couI)] = srcData[idx];
                destData[(couJ * (width * 2) + (couI + 1))] = srcData[idx];
                destData[(couJ + 1) * (width * 2) + couI] = srcData[idx];
                destData[(couJ + 1) * (width * 2) + (couI + 1)] = srcData[idx];
                couI += 2;
            }
            couJ += 2;
        }

        return filterBartlett(dest);
    }
}
```

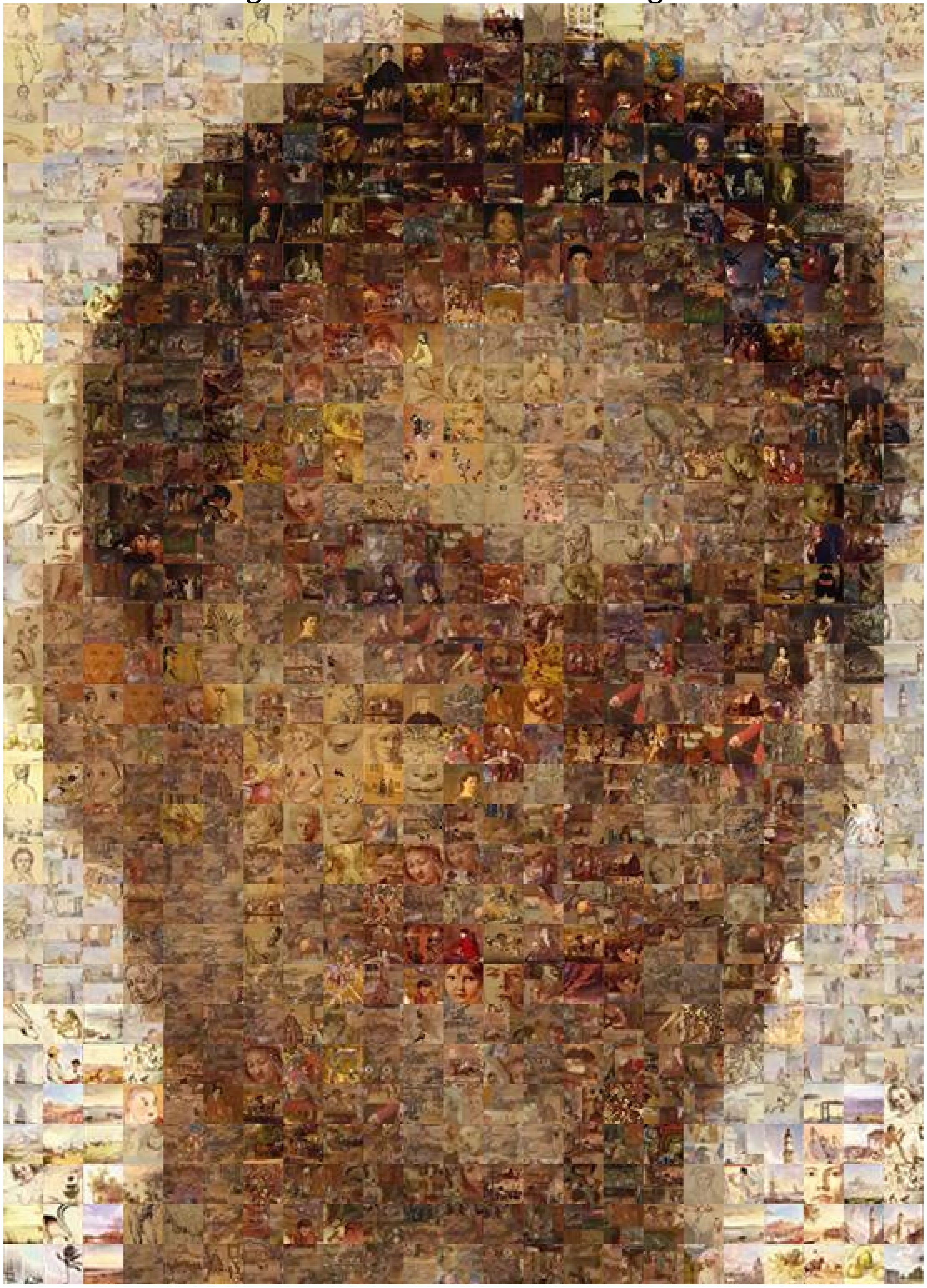
11.25 PGUtil

```
package photogram;

import java.io.File;
import java.util.StringTokenizer;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: Photogram LLC</p>
 * @author Ohan Oda
 * @version 1.0
 */
public class PGUtil {
    /**
     * Get the extention of a file.
     * @param f File
     * @return String
     */
    public static String getExtension(File f) {
        String s = f.getName();
        return getExtension(s);
    }

    public static String getExtension(String s){
        String ext = "";
        int i = s.lastIndexOf('.');

        if (i > 0 && i < s.length() - 1) {
            ext = s.substring(i + 1).toLowerCase();
        }
        return ext;
    }

    public static String removeExtension(File f) {
        String s = f.getName();
        return removeExtension(s);
    }

    public static String removeExtension(String s){
        StringTokenizer st = new StringTokenizer(s, ".");
        return st.nextToken();
    }

    public static int getFrontTabCount(String s){
        int count = 0;
        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) == '\t')
                count++;
            else
                break;
        }

        return count;
    }

    public static double Binomial(int n, int s)
    {
        double res = 1;
        for (int i = 1 ; i <= s ; i++)
            res = (n - i + 1) * res / i ;
        return res;
    }
}
```

11.26 Photomontage created with 16x16 size images



11.27 Photomontage created with 24x24 size images



11.28 Photomontage created with 32x32 size images