

Photon

PLT - Spring 2021

Akira Higaki (abh2171) - Manager

Calum McCartan (cm4114) - System Architect

Franky Campuzano (fc2608) - Language Guru

Phu Pham (pdp2121) - Tester



April 26, 2021

Contents

1	Introduction	4
2	Language Tutorial	4
2.1	Compiling and Running Photon	4
2.2	Simple Image Example	5
3	Language Reference Manual	6
3.1	Data Types	6
3.1.1	Primitives	6
3.1.2	Pint Type	6
3.1.3	Structures	7
3.2	Automatic Type Conversion	7
3.3	Lexical Conventions	7
3.3.1	Identifiers	7
3.4	Keywords	7
3.5	Operators & Logical Expressions	8
3.6	Precedence	8
3.7	Syntax	9
3.7.1	Variable Declaration & Assignment	9
3.7.2	Variable Hoisting	9
3.7.3	Naming	10
3.7.4	String literals	10
3.7.5	Comments	10
3.7.6	White space	10
3.8	Pixel	10
3.9	Image	11
3.9.1	Creating an image	11
3.9.2	Width and height	11
3.9.3	Accessing and setting a pixel	11
3.9.4	Image functions	11
3.9.5	Adding Images	11
3.9.6	Subtracting Images	12
3.9.7	Loading an Image	12
3.9.8	Saving an Image	12
3.9.9	Inverting an image	13
3.9.10	Pasting an image on top of another image	13
3.9.11	Grayscale-ing an image	13
3.9.12	Flipping an image	13
3.10	Attributes	13
3.11	Arrays	14
3.11.1	Initialization	14
3.11.2	Size	14
3.11.3	Element Retrieval	14
3.11.4	Length	14

3.12	Functions	14
3.13	Control Flow	15
3.14	Standard Library Functions	15
	3.14.1 Min, Max Functions	16
	3.14.2 Printing	16
3.15	Destroy	16
	3.15.1 Image and Pixel Functions	16
3.16	Example Code	17
	3.16.1 Maximum of an array	17
	3.16.2 Example Image Modification Flow	17
	3.16.3 Image addition	17
4	Project Plan	18
4.1	Planning Process	18
4.2	Specification Process	18
4.3	Development Process	18
4.4	Testing Process	19
4.5	Programming Style	19
4.6	Project Timeline	19
4.7	Team Member Roles	19
4.8	Software Environment	20
4.9	Project Log	20
5	Architectural Design	22
5.1	Compiler Diagram	22
5.2	Scanner	23
5.3	Parser	23
5.4	Semantic Checking	23
5.5	Code Generation	24
5.6	Linking & the Executable	24
5.7	Module Contribution	24
6	Language Evolution	25
6.1	Photon: At the Beginning	25
6.2	Rethinking Our Design Choices	25
6.3	Photon: Present-day	25
7	Test Plan	26
7.1	Source and Target Language Programs	26
7.2	Test Cases	29
7.3	Test Suites & Automation	29
7.4	Test Roles	32

8	Lessons Learned	32
8.1	Akira	32
8.2	Calum	33
8.3	Franky	33
8.4	Phu	34
9	Future Improvements	35
9.1	Extra Features	35
9.2	Runtime Error Handling	35
10	Acknowledgements and References	35
11	Appendix	36
11.1	Code Listing	36
11.1.1	photon.ml	36
11.1.2	scanner.mll	37
11.1.3	ast.ml	38
11.1.4	photonparse.mly	41
11.1.5	sast.ml	44
11.1.6	semant.ml	47
11.1.7	codegen.ml	54
11.1.8	Image.c	68
11.1.9	Image.h	76
11.1.10	utils.c	77
11.1.11	utils.h	78
11.1.12	Makefile	78
11.1.13	testall.sh	79
11.1.14	Sample Images	85
11.2	Test Scripts	86

1 Introduction

Photon is a language that is centered around modifying and editing images, similar to the functionality of Adobe Photoshop or Adobe After Effects. The language is inspired by workflows in the visual effects industry, especially the node-based software Nuke. It aims to be able to provide functionality similar to that of Nuke through a C-like syntax.

The main feature of the language is that takes advantage of an alpha layer in addition to the red, green, and blue layers to provide an efficient and easy way to combine and edit images and video. An alpha layer is a fourth layer that governs the transparency of a pixel. By modifying the alpha value in select pixels, the user is able to create modified images easily.

In the visual effects industry, there are a lot of simple procedures (such as greenscreen) that are simple but tedious to do by hand, especially since people must modify each frame at a time. Photon brings a solution to the problem by offering an efficient way to output editing procedures quickly.

The end goal is to have users upload PNG images (or multiple PNGs for videos) and put it through an automated and efficient editing pipeline through our language to output simple editing procedures quickly.

2 Language Tutorial

All of the required files, including the C library used to interface with images, are included in the source files. Make sure Ocaml and LLVM are installed on your system.

2.1 Compiling and Running Photon

Here is an example program that we can run named test-hello.phn

```
1 func int main()
2 {
3     print(42);
4     print(71);
5     print(1);
6     return 0;
7 }
```

Note that there is a mandatory main function that has no parameters which returns an integer. This test calls the built in print function, which takes an integer and prints it on a new line. Lastly, the program returns 0.

To compile this test file, see below.

```
1 make all
2 ./photon.native test-hello.phn > test-hello.ll
```

```
3 llc -relocation-model=pic test-hello.ll > test-hello.s
4 cc -o test-hello.exe test-hello.s utils.o Image.o -lm
```

This should give the output:

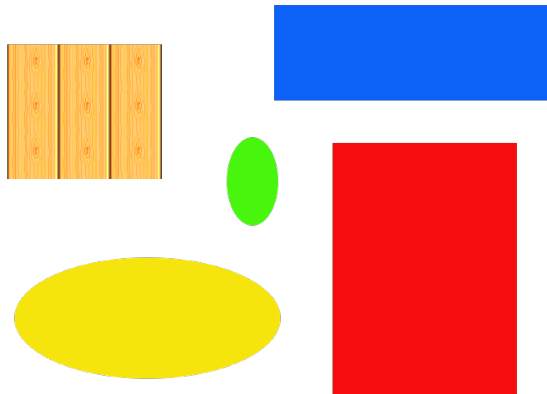
```
1 ./test-hello.exe
2 42
3 71
4 1
```

2.2 Simple Image Example

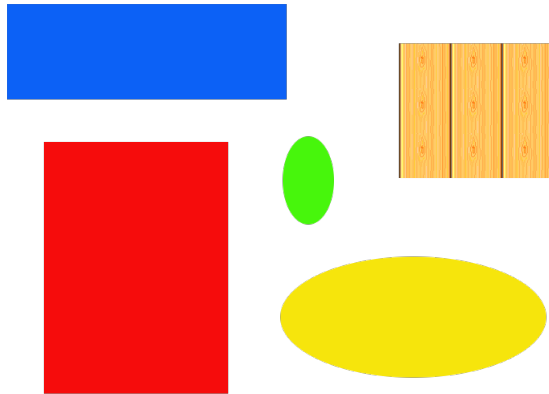
Here is a very simple example of how to edit an image using a built in function.

```
1 func int main()
2 {
3     Image img;
4     Image flippedimg;
5
6     img = load("Shapes.png");
7     flippedimg = flip(img);
8     save(flippedimg, "flipImgTest.png");
9
10    return 0;
11 }
```

Note that you must use a PNG file. The image file must be in the same directory as the .exe file. This program outputs a flipped image, flipped horizontally. Shapes.png:



flipImgTest.png:



Refer to the language reference manual for all built in functions and other uses.

3 Language Reference Manual

3.1 Data Types

3.1.1 Primitives

Type	Description
int	An integer
float	A floating-point number
string	A sequence of characters
bool	True or False values
pint	A special type of Int whose value can only range from 0 to 255

3.1.2 Pint Type

The unsigned 8-bit pint type can be used to efficiently store the four RGBA values of a pixel in the space of a single 32-bit number. In addition to this, overflow is automatically prevented on the pint type by clamping the values between 0 and 255 instead. This is very useful for many of the operations that are commonly performed when manipulating pixels.

```

1 pint p;
2 pint q;
3 p = 150;
4 q = 200;
5 print(p + q); # prints 255
6 print(p - q); # prints 0

```

3.1.3 Structures

Type	Description
array	A single unit of multiple grouped values
Pixel	A group of four Pint values
Image	A one dimensional array of pixel values.

3.2 Automatic Type Conversion

Photon does not support explicit type conversion through the use of built-in functions. However, if you attempt to perform, for example, an arithmetic operation involving different numeric primitive data types (float, int, pint), then Photon will automatically convert to the most specific data type.

Values for numeric types are also automatically cast when assigned, returned or used as arguments for some combinations. The supported conversions are listed below.

Required type	Accepted types
int	int, pint
pint	int, pint
float	int, pint, float

3.3 Lexical Conventions

3.3.1 Identifiers

User-defined variables and functions must have a letter as the first character, followed by any letter, number, or `_`. eg. `edit_Image()`.

3.4 Keywords

These are the keywords that are reserved by the language for conditional statements and function declarations.

Name	Description
<code>func</code>	Function declaration
<code>return</code>	Followed by a value that is returned to the caller
<code>if</code>	Beginning of conditional statement
<code>else</code>	Conditional statement
<code>for</code>	Iterative statement
<code>while</code>	Iterative statement

Below are keywords reserved for data types and structures in the language.

Name
int
float
string
bool
pint
array
Pixel
Image

Color aliases are also keywords reserved by the language - which can be referenced using a '_', such as _black

Alias	Value
_black	Pixel(0, 0, 0, 255)
_white	Pixel(255, 255, 255, 255)
_gray	Pixel(128, 128, 128, 255)
_red	Pixel(255, 0, 0, 255)
_green	Pixel(0, 255, 0, 255)
_blue	Pixel(0, 0, 255, 255)
_cyan	Pixel(0, 255, 255, 255)
_magenta	Pixel(255, 0, 255, 255)
_yellow	Pixel(255, 255, 0, 255)

3.5 Operators & Logical Expressions

Type	Description
+	Adds primitives
-	Subtracts primitives
*	Multiplies primitives
/	Divides non-zero primitives
==, <=, >=, <, >	Compares primitives
[]	Array creation and element calling
sqrt(arg)	Square root of a numeric type (outputs a float)
max(arg1, arg2)	Maximum between arg1 and arg2 (integers only, outputs an int)
min(arg1, arg2)	Minimum between arg1 and arg2 (integers only, outputs an int)
	OR operator
&&	AND operator
!	NOT operator

3.6 Precedence

Precedence	Operator	Description	Associativity
1	() [] .	Function call Array subscripting Structure and union member access	Left-to-right
2	!=	Logical NOT	Right-to-left
3	sqrt() min() max()	Square root, min, and max	Right-to-left
4	* /	Multiplication, division	Left-to-right
5	+ -	Addition and subtraction	Left-to-right
7	< <= > >=	For relational operators < and <= respectively For relational operators > and >= respectively	Left-to-right
8	==	For relational =	Left-to-right
9	&	Logical AND	Left-to-right
10		Logical OR	Left-to-right
11	=	Simple assignment	Left-to-right

3.7 Syntax

3.7.1 Variable Declaration & Assignment

All types of variable can be declared, assigned and reassigned in the same way.

```

1 <type> <name>;           # Declaration
2 <name> = <value>;       # Assignment / reassignment

```

For example...

```

1 int x;
2 x = 42;
3 int y;
4 bool myBool = true;
5 y = x + 10;
6 x = 3;
7 string greeting;
8 greeting = "Hello";

```

All variables declared without assignment will be assigned a default value. 0 for numeric types, false for the boolean type, and "" for string type.

3.7.2 Variable Hoisting

Variables are hoisted (like in JavaScript), which means they can be used before they are declared, just as long as they are still declared somewhere in the current scope. This means that the following is valid.

```

1 x = 42;
2 int x;

```

3.7.3 Naming

Variables and functions must be named any character a-z, followed by any number of characters a-z, digits 0-9, or underscores '_' (Characters may be upper or lower case). Variables and functions may not use reserved words as names.

```
1 int GoodName123;  
2 float other321_name;
```

3.7.4 String literals

String literals are any number of ASCII characters enclosed in a pair of double quotes. Strings may not include the double quote character.

```
1 string x;  
2 x = "hello `friend`";  
3 prints("hi there");
```

3.7.5 Comments

Line comments are denoted by the hash character '#'. Hash characters inside string literals are unaffected.

3.7.6 White space

Comments are terminated by newline characters. Other than this, the language is not sensitive to white space / indentation.

3.8 Pixel

The 32-bit pixel type can be created with the pixel function which takes 4 pint arguments corresponding to RGBA values in the range of 0-255.

```
1 Pixel myPurplePixel;  
2 myPurplePixel = Pixel(150, 20, 200, 255);
```

Pixels can also be created using the aliases specified in the keywords section. Example uses of the aliases are shown below.

```
1 set_pixel(img, 10, 4, _green);  
2 favouriteColor = _magenta;
```

The RGBA values of a pixel can be accessed as attributes which each return a pint type.

```
1 x = myPixel.r;  
2 y = myPixel.g;
```

Unlike the Image type, Pixels are passed by value rather than by reference and so do not need to be manually destroyed.

3.9 Image

Images are structs which hold a one-dimensional array of red, green, blue, and alpha-layer values.

3.9.1 Creating an image

The image type can be created with a width, height, and background color.

```
1 Image img;
2 img = create(600, 400, _blue);
```

3.9.2 Width and height

The width and height attribute is accessible.

```
1 wid = width(img); #both functions return an int
2 ht = height(img);
```

3.9.3 Accessing and setting a pixel

Pixels in an image can be accessed and set like so.

```
1 favouritePixel = get_pixel(img, 12, 18);
2 set_pixel(myImg, 22, 27, favouritePixel);
```

3.9.4 Image functions

Images can be manipulated with several built-in functions, including flip, to_gray, image_invert:

```
1 myImg = load("myImage.png");
2 flip(myImg);
3 to_grey(myImg);
4 image_invert(myImg);
```

3.9.5 Adding Images

Two images can be combined with the + or - operators. Below is the code snippet for adding two images:

```
1 newImg = img1 + img2;
```

Images are added with the following equation:

```
1 newImg.pixels[x][y].red =
2 img1.pixels[x][y].red * img1.pixel[x][y].alpha / 255 +
3 img2.pixels[x][y].red * img2.pixel[x][y].alpha / 255
```

This operation occurs with all pixels in both images. Adding two images of different size will result in the produced image being the maximum height and width of the two images, and the addition will be aligned to the top left corner of both images. Keep in mind that the assigned value is a pixel with a values bound between 0 and 255.

3.9.6 Subtracting Images

Below is the code snippet for subtracting two images:

```
1 newImg = img1 - img2;
```

Images are subtracted with the following equation:

```
1 newImg.pixel[x][y].red =
2 img1.pixel[x][y].red * img1.pixel[x][y].alpha / 255 -
3 img2.pixel[x][y].red * img2.pixel[x][y].alpha / 255
```

This operation occurs with all pixels in both images. Subtracting two images of different size will result in the resulting image being the maximum height and width of the two images, and the subtraction will be aligned to the top left corner of both images. Keep in mind that the assigned value is a pixel with a maximum value of 255 and a minimum value of 0.

3.9.7 Loading an Image

A new image is loaded with the built-in function like so:

```
1 Image myImg;
2 myImg = load(filePath);
```

Here, filePath is the name of a file in the same directory. The image being loaded must be a .png file, must have a bit-depth of 8 or 32, and it must contain four channels: red, green, blue, and alpha-layer values. You can use an online image converter to avoid these problems.

3.9.8 Saving an Image

Modification to an image is saved to a file system with the built-in function like so:

```
1 Image myImg;
```

```
2 myImg = load(filePath);
3 save(myImg, filename);
```

Here, filename is a string containing desired name of the file. It will be saved into the current directory.

3.9.9 Inverting an image

You can invert an image using the built-in `image_invert()` function. You can then save this modification using the `save()` function.

```
1 inverted_image = image_invert(oldimg);
```

3.9.10 Pasting an image on top of another image

You can paste an image at a desired width, height coordinate using `image_paste()`. You can then save this image using `save()`.

```
1 paste_image = image_paste(img_target, img_source, 0, 0);
```

3.9.11 Grayscale-ing an image

Using the `to_gray()` function, you can create a grayscale copy of an image.

```
1 grayimg = to_gray(oldimg);
```

3.9.12 Flipping an image

You can flip an image using the built-in `flip()` function. You can then save this modification using the `save()` function.

```
1 flipped_image = flip(oldimg);
```

3.10 Attributes

Attributes of certain data types - array, Image and Pixel - can be accessed using the name of variable and the name of the attribute separated by a dot.

```
1 len = myArray.length;
2 redValue = pixel.r;
3 w = img1.width;
```

3.11 Arrays

3.11.1 Initialization

Arrays are a collection of items - all of which must be of the same data type. They are one-dimensional.

Arrays are a non-essential component of Photon, and AP++, a Fall 2018 project, served as a guide on how to implement arrays. However, arrays exist in the language to provide further functionality to the users of Photon. For more details about their implementation, read the Acknowledgements and References section down below.

```
1 string [] greetings;
```

3.11.2 Size

Arrays can be created with specified values, meaning their size is implicit. You can also add elements, one at a time to the array, meaning their size is also dynamic.

```
1 greetings = ["hello", "hi"]; # size is implicit
2 array_add(greetings, "welcome"); #size is dynamic
```

3.11.3 Element Retrieval

Array values are assigned and retrieved like so.

```
1 prints(greetings[1]); #prints "hi"
```

3.11.4 Length

Array size are retrieved using the length keyword.

```
1 greetings = ["hello", "hi"];
2 print(greetings.length); #output 2
```

3.12 Functions

Functions are declared using the following syntax.

```
1 func <return_type> <name>(<arg1type> <arg1name>) {
2     return <value>;
3 }
```

Function arguments are pass by value.
For example a function can be declared and called like so.

```
1 func int add(int val1, int val2) {
2     return val1 + val2;
3 }
4
5 #somewhere in main...
6 int sum = add(3, 5);
```

More function examples included under Example Code.

3.13 Control Flow

Control flow statements ignore white space. Scope is defined by the usage of {} brackets.

Conditional blocks can be created as shown below.

```
1     if(conditional statement is true){
2         #do this
3     }
4
5     #using blocks
6     if(x>0) {
7         prints("x is positive");
8     }
9     elif(x<0) {
10        prints("x is negative");
11    }
12    else { prints("x equals 0"); }
```

Loops are created as shown below.

```
1     for( optional value initialization; conditional
2     statement; optional increment) {
3         #do stuff
4     }
5
6     while(conditional statement is true) {
7         #do stuff
8     }
```

3.14 Standard Library Functions

Built-in functions in Photon

3.14.1 Min, Max Functions

Min() and max() functions each take two ints as arguments, and return the smallest or largest value, respectively.

```
1     int x = min(int_one , int_two);
2     int y = max(int_one , int_two);
```

3.14.2 Printing

Printing in Photon is typed - meaning there are different functions for each primitive data type. Pints and ints share the same printing function.

```
1     print(int);
2     print(pint);
3     prints(string);
4     printf(float);
5     printb(bool);
6     print(array[element of type int]);
```

3.15 Destroy

Images are passed by reference, and so need to be manually destroyed in Photon calling. This frees the memory allocated to the image in the heap.

```
1     Image img;
2     img = load("edwards.png");
3     #do some stuff to edwards
4     destroy(img);
```

3.15.1 Image and Pixel Functions

Image and Pixel functions are built into the language. Detailed descriptions of these functions are listed above in the Image and Pixel sections.

The following serves as a quick reference sheet for Image and Pixel functions.

```
1     Image load(string filename);
2     void save(Image img, string filename);
3     Image create( Int width, Int height, Pixel p);
4
5     int width(Image img);
6     int height(Image img);
7
8     Pixel get_pixel(Image img, int x, int y);
9     int set_pixel(Image img, int x, int y, Pixel p);
10
```

```

11     Image image_add(Image img1, Image img2);
12     Image image_subtract(Image img1, Image img2);
13
14     Image image_invert(Image orig);
15     Image image_paste(Image target, Image source, x, y);
16     Image to_gray(Image orig);
17     Image flip(Image orig);

```

3.16 Example Code

3.16.1 Maximum of an array

This is a simple subroutine that finds the largest element in an array.

```

1 func int maxElement(int[] inArray) {
2     int max;
3     max = 0;
4     int i;
5     for (i = 0; i < inArray.length; i = i + 1) {
6         if (inArray[i] > max) {
7             max = inArray[i];
8         }
9     }
10    return max;
11 }

```

3.16.2 Example Image Modification Flow

Below is a subroutine that uses the built-in functions to modifying an image. This demonstrates loading/saving in an image.

```

1 func string flipAndGreyImage(string filePath) {
2     Image testImage;
3     testImage = load(filePath);
4     testImage = flip(testImage);
5     testImage = to_gray(testImage);
6     string retPath;
7     retPath = "newImage.png";
8     Image newImage;
9     save(testImage, retPath);
10    return retPath;
11 }

```

3.16.3 Image addition

Below is a subroutine that uses the alpha values to combine two images together. This works best with equal sized images.

```
1 func Image halfHalf(Image image1, Image image2) { # alphas
    = 255 by default
2     int i;
3     int j;
4     Pixel blankp;
5     blankp = pixel(0, 0, 0, 0);
6     for (i = 0; i < width(image1)/2; i = i + 1) {
7         for (j = 0; j < width(image2); j = j + 1) {
8             set_pixel(image1, i, j, blankp);
9             set_pixel(image2, i+image1.width/2, j, blankp);
10        }
11    }
12    Image image3;
13    image3 = image1 + image2;
14    return image3; #left side of image1, right side of
    image 2
15 }
```

4 Project Plan

4.1 Planning Process

For each of the milestones placed by Professor Edwards, we created smaller goal for each person to complete. These goals would be scaled to the weekly meetings that we would hold. We used a Trello board which kept track of each task and who did what to enable transparency. If someone was having difficulty with their specific goal, we would brainstorm solutions together in our meetings as well as have other teammates help if needed.

4.2 Specification Process

From the submission of the proposal, we already have decided to create a C-like language in terms of syntax. We further specified our intended syntax in the submission of the LRM. As we created the actual language, we saw that some of our specifications were not necessary, and some were. In addition, while working with test cases, we found that we needed more built-in functions than originally expected due to the image-based application. Before every change to the LRM specifications, we would discuss as a group what the specifications would entail and make sure everyone was in agreement.

4.3 Development Process

Our development of the compiler followed the milestones set by Professor Edwards. The lexer and parser were written first, then the semantic checker, then

the code generator. We found that breaking the work into pieces of functionality within the language worked best.

4.4 Testing Process

Within every feature that we created, we added at least one unit test that tested its functionality. To pinpoint specific bugs, we restricted tests by one or two functionalities. Since we are interfacing with a C library, we also did tests only in the C compiler to test functionality within our use of C. Using the testing log and llvm output were helpful in finding bugs in our code.

4.5 Programming Style

Our team generally followed the Ocaml and C formatting styles.

- In Ocaml files, indentations are 2 spaces wide.
- In C and Photon files, indentations are 4 spaces.
- Block comments are on top of every file to denote functionality and authors
- Comments are also placed to break up long files such as codegen.ml
- Built-in functions use underscore to separate words

4.6 Project Timeline

Date	Milestone
February 3	Language proposal submitted
February 16	Git repo created, first commit
February 24	LRM and Parser finished
March 24	First successfully generated code
April 25	Photon compiler finished
April 26	Final report finished

4.7 Team Member Roles

These responsibilities did not strictly dictate what people worked on. The vast majority of the work done was overlapping, however the list below highlights the small differences in responsibilities.

Team Member	Responsibilities
Akira Higaki	Manager, C library integration
Calum McCartan	Semantic Checking, Special Types
Franky Campuzano	Built-in functions, Presentation Slides
Phu Pham	Testing

4.8 Software Environment

We used the following software environment:

- Ocaml 4.05.0
- Github for version control
- CC for building the exe as well as linking to the C libraries
- VSCode for file editing

4.9 Project Log

This is the full commit history of the project.

```
Sun Apr 25 22:03:54 2021 -0500 - Franky : updated image_paste to work with an x,y position.
Sun Apr 25 21:41:54 2021 -0400 - Akira : Moved readme out of /code
Sun Apr 25 21:41:07 2021 -0400 - Akira : Updated README
Sun Apr 25 20:04:51 2021 -0400 - CalumMcCartan : Update makefile and _tags
Sun Apr 25 19:44:50 2021 -0400 - CalumMcCartan : Merge branch 'main' of https://github.com/CalumMcCartan/Photon
into main
Sun Apr 25 19:44:41 2021 -0400 - CalumMcCartan : testall.sh tweak
Sun Apr 25 18:16:37 2021 -0500 - Franky : Added image_invert.
Sun Apr 25 19:02:23 2021 -0400 - Akira : Removed printbig
Sun Apr 25 18:38:38 2021 -0400 - CalumMcCartan : Merge branch 'main' of https://github.com/CalumMcCartan/Photon
into main
Sun Apr 25 18:38:24 2021 -0400 - CalumMcCartan : Add authors to heading
Sun Apr 25 17:35:52 2021 -0500 - Franky : Added reference for arrays.
Sun Apr 25 17:34:31 2021 -0500 - Franky : Small fix to set pixel.
Sun Apr 25 18:27:46 2021 -0400 - Phu Pham : bug fixes
Sun Apr 25 18:22:15 2021 -0400 - Phu Pham : Merge branch 'main' of github.com:CalumMcCartan/Photon
into main
Sun Apr 25 18:20:34 2021 -0400 - Phu Pham : tests added
Sun Apr 25 17:40:16 2021 -0400 - CalumMcCartan : Delete old version of Photon
Sun Apr 25 17:35:26 2021 -0400 - CalumMcCartan : Merge branch 'main' of https://github.com/CalumMcCartan/Photon
into main
Sun Apr 25 17:35:15 2021 -0400 - CalumMcCartan : create now uses pixel
Sun Apr 25 17:33:23 2021 -0400 - Akira : Added header comments on C files
Sun Apr 25 17:19:06 2021 -0400 - CalumMcCartan : Image operators
Sun Apr 25 16:42:14 2021 -0400 - CalumMcCartan : width and height attrs and tests
Sun Apr 25 16:19:35 2021 -0400 - CalumMcCartan : Make pixels pass by value
Sun Apr 25 15:54:35 2021 -0400 - CalumMcCartan : fix getpixel test
Sun Apr 25 15:51:06 2021 -0400 - CalumMcCartan : Add pixel type
Sun Apr 25 13:37:22 2021 -0400 - CalumMcCartan : use input and output folders for images
Sun Apr 25 12:01:11 2021 -0500 - Franky : Added get_position helper function.
Sun Apr 25 12:48:03 2021 -0400 - Phu Pham : subtract image added
Sun Apr 25 12:02:35 2021 -0400 - Phu Pham : merge conflicts resolved
Sun Apr 25 11:56:25 2021 -0400 - Phu Pham : image-add added
Sun Apr 25 11:15:30 2021 -0400 - CalumMcCartan : Simplify func definitions
Sun Apr 25 10:17:40 2021 -0400 - CalumMcCartan : simplify function args
Sun Apr 25 02:18:38 2021 -0500 - Franky : Added image_paste(img1, img2). Places img2 on top of img1.
Sun Apr 25 02:30:35 2021 -0400 - Akira : Added flip function
Sun Apr 25 00:10:32 2021 -0400 - Akira : Added to_gray function
Sat Apr 24 23:48:54 2021 -0400 - Akira : Added image destroy built-in function
Sat Apr 24 23:32:08 2021 -0400 - Akira : Merge branch 'main' of https://github.com/CalumMcCartan/Photon
Sat Apr 24 23:29:29 2021 -0400 - Akira : Added image create
Sat Apr 24 22:25:29 2021 -0500 - Franky : Added set_pixel. Only changes a single pixel.
Sat Apr 24 22:35:25 2021 -0400 - CalumMcCartan : Image saving
Sat Apr 24 21:08:27 2021 -0400 - CalumMcCartan : Prevent pint overflow
Sat Apr 24 17:40:47 2021 -0500 - Franky : get_pixel. you can specify the position in the image you want,
prints r,g,b,a values.
Sat Apr 24 18:39:43 2021 -0400 - Phu Pham : rename printbig to utils
Sat Apr 24 18:24:55 2021 -0400 - Phu Pham : sqrt added with tests
Sat Apr 24 17:18:50 2021 -0400 - Akira : Added changes to Image.c to force an alpha layer in an image
```

Sat Apr 24 15:44:55 2021 -0500 - Franky : Testing get_pixel.
 Sat Apr 24 16:00:50 2021 -0400 - CalumMcCartan : add pixel alias
 Sat Apr 24 15:39:01 2021 -0400 - CalumMcCartan : Colour aliases
 Sat Apr 24 15:04:56 2021 -0400 - CalumMcCartan : add _red colour alias
 Sat Apr 24 13:23:15 2021 -0400 - CalumMcCartan : hoist variable decls
 Sat Apr 24 02:59:29 2021 -0500 - Franky : Added get_pixel function and one test.
 Sat Apr 24 01:37:47 2021 -0500 - Franky : More small changes to arrays. Formatting for codegen.
 Sat Apr 24 01:09:43 2021 -0500 - Franky : Small change to arrays.
 Sat Apr 24 01:57:56 2021 -0400 - CalumMcCartan : img width and height funcs
 Fri Apr 23 14:19:42 2021 -0400 - CalumMcCartan : Auto cast func args (and simplify func codegen)
 Fri Apr 23 11:18:21 2021 -0400 - CalumMcCartan : Automatic return type casting
 Fri Apr 23 02:59:04 2021 -0400 - Calum McCartan : Merge pull request #3 from CalumMcCartan/pints
 Fri Apr 23 02:39:21 2021 -0400 - CalumMcCartan : cleanup casting
 Fri Apr 23 02:16:56 2021 -0400 - CalumMcCartan : Auto casting for numeric binops
 Thu Apr 22 23:23:00 2021 -0400 - CalumMcCartan : automatic numeric assignment casting
 Thu Apr 22 22:26:45 2021 -0400 - CalumMcCartan : Convert int to pint during pint assignment
 Thu Apr 22 16:52:27 2021 -0400 - CalumMcCartan : Merge branch 'pints' of <https://github.com/CalumMcCartan/Photon>
 into pints
 Wed Apr 21 09:19:12 2021 -0400 - CalumMcCartan : Pint convert test
 Wed Apr 7 11:23:38 2021 -0400 - CalumMcCartan : Add pint, allow to take int value
 Thu Apr 22 16:44:31 2021 -0400 - CalumMcCartan : indentation cleanup
 Thu Apr 22 16:40:37 2021 -0400 - CalumMcCartan : Slightly better indentation
 Thu Apr 22 11:13:14 2021 -0400 - CalumMcCartan : Merge branch 'pints' of <https://github.com/CalumMcCartan/Photon>
 into pints
 Wed Apr 21 09:19:12 2021 -0400 - CalumMcCartan : Pint convert test
 Wed Apr 7 11:23:38 2021 -0400 - CalumMcCartan : Add pint, allow to take int value
 Thu Apr 22 11:04:56 2021 -0400 - CalumMcCartan : Allow binop between int and float literals
 Thu Apr 22 04:26:32 2021 -0400 - Akira : Added image type (llvm pointer type) and image load function
 Wed Apr 21 09:21:25 2021 -0400 - CalumMcCartan : merge
 Wed Apr 21 09:19:12 2021 -0400 - CalumMcCartan : Pint convert test
 Wed Apr 7 11:23:38 2021 -0400 - CalumMcCartan : Add pint, allow to take int value
 Tue Apr 20 14:21:34 2021 -0500 - Franky : Added array test for length and for loops.
 Mon Apr 19 23:20:15 2021 -0500 - Franky : Added arrays. Works for floats, ints, and arrays. 4 tests added.
 Thu Apr 15 18:00:11 2021 -0400 - Phu Pham : small fixes
 Thu Apr 15 17:41:44 2021 -0400 - Phu Pham : min max built in added
 Mon Apr 12 14:50:33 2021 -0400 - Akira : Added makefile for C image testing
 Mon Apr 12 14:38:04 2021 -0400 - Akira : Added stb_image library for C and example way to use it
 (no changes to Photon)
 Wed Apr 7 11:23:38 2021 -0400 - CalumMcCartan : Add pint, allow to take int value
 Tue Apr 6 00:27:53 2021 -0400 - Akira : Added 'func' for function declaration and changed all of the tests
 to work
 Mon Apr 5 12:21:52 2021 -0400 - Phu Pham : line comment added
 Mon Apr 5 00:33:08 2021 -0500 - Franky : Added strings and printing strings.
 Tue Mar 30 17:28:05 2021 -0400 - Calum McCartan : Merge pull request #2 from CalumMcCartan/calum/hello-world
 Tue Mar 30 17:27:39 2021 -0400 - CalumMcCartan : Re-restart from microc
 Wed Mar 24 18:51:35 2021 -0400 - Akira : Deleted failed tests, edited README
 Wed Mar 24 18:45:42 2021 -0400 - Akira : Added variable assignment in parser
 Wed Mar 24 16:27:28 2021 -0500 - Franky : Added print functionality.
 Wed Mar 24 16:38:03 2021 -0400 - CalumMcCartan : Add int expression
 Wed Mar 24 15:29:14 2021 -0400 - CalumMcCartan : Fix global variable dec
 Wed Mar 24 14:09:41 2021 -0400 - CalumMcCartan : Can declare (empty) functions
 Wed Mar 24 11:56:50 2021 -0400 - CalumMcCartan : compiles with photon's scanner
 Tue Mar 23 19:17:11 2021 -0400 - CalumMcCartan : Use photons parser, scanner, ast
 Tue Mar 23 19:14:13 2021 -0400 - Calum McCartan : Merge pull request #1 from CalumMcCartan/start-from-microc
 Tue Mar 23 18:31:03 2021 -0400 - CalumMcCartan : Use microc as a starting point
 Tue Mar 23 00:55:13 2021 -0500 - Franky : Merge branch 'main' of <https://github.com/CalumMcCartan/Photon>
 Tue Mar 23 00:54:57 2021 -0500 - Franky : Needed by running make on codegen.
 Tue Mar 23 01:30:12 2021 -0400 - Akira : Merge branch 'main' of <https://github.com/CalumMcCartan/Photon>
 Tue Mar 23 01:30:01 2021 -0400 - Akira : Added "type varname = expr" to parser
 Mon Mar 22 22:49:49 2021 -0500 - Franky : Fixed typo. Also testing branches, idk how they work.
 Mon Mar 22 22:44:12 2021 -0500 - Franky : super super rough version of codegen. does not work, and has
 not been tested. requires changing photon.ml file.
 Mon Mar 22 22:41:38 2021 -0500 - Franky : Sample hello world file.
 Mon Mar 22 22:41:01 2021 -0500 - Franky : Added run line to test a sample hello world file.
 Mon Mar 22 13:14:11 2021 -0400 - Akira : Merge branch 'main' of <https://github.com/CalumMcCartan/Photon>
 Mon Mar 22 13:13:51 2021 -0400 - Akira : Added sast.ml
 Fri Mar 19 09:33:54 2021 -0400 - CalumMcCartan : fix non exhaustive matching warning
 Thu Mar 18 15:56:27 2021 -0400 - Akira : Fixed function parsing and punctuation bugs

```

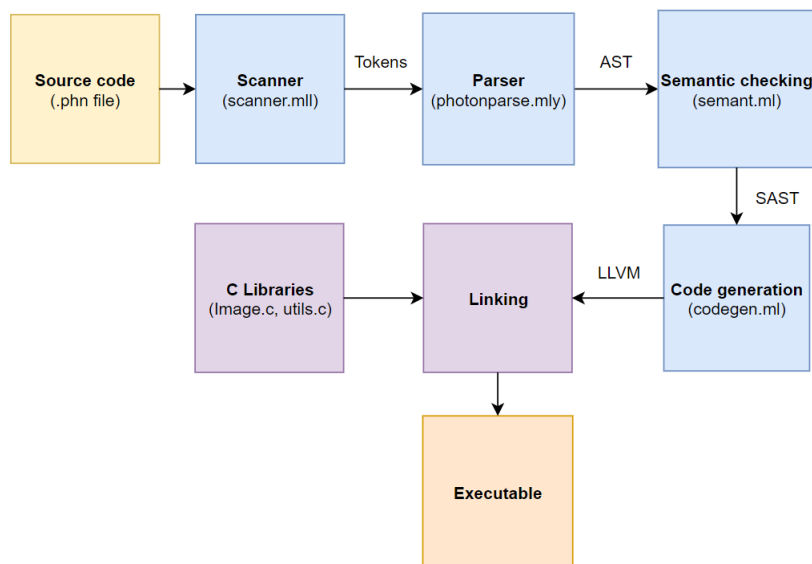
Wed Feb 24 21:15:15 2021 -0500 - CalumMcCartan : Added null and func arguemnts
Wed Feb 24 21:00:09 2021 -0500 - CalumMcCartan : Started string literal
Wed Feb 24 20:33:44 2021 -0500 - CalumMcCartan : Add comments and more general entry point
Wed Feb 24 17:42:49 2021 -0500 - CalumMcCartan : Array literals
Wed Feb 24 16:20:11 2021 -0500 - CalumMcCartan : Add negaiton
Wed Feb 24 15:56:06 2021 -0500 - CalumMcCartan : Start on arrays
Wed Feb 24 15:37:26 2021 -0500 - Akira Higaki : fix statements bug
Wed Feb 24 15:27:27 2021 -0500 - Akira Higaki : Small bug fix
Wed Feb 24 15:26:04 2021 -0500 - Akira Higaki : Merge branch 'main' of https://github.com/CalumMcCartan/Photon
Wed Feb 24 15:22:03 2021 -0500 - Akira Higaki : Added loops, if, and colors
Wed Feb 24 14:01:05 2021 -0500 - CalumMcCartan : dont attach string to data type names
Wed Feb 24 11:03:18 2021 -0600 - Franky : Added "." function calls.
Wed Feb 24 11:33:09 2021 -0500 - CalumMcCartan : Fix shift errors by fixing semicolons
Tue Feb 23 22:37:30 2021 -0500 - Phu Pham : data type added, works
Tue Feb 23 14:01:42 2021 -0500 - Phu Pham : data type added, conflict resolved
Tue Feb 23 13:57:44 2021 -0500 - Phu Pham : data type added
Tue Feb 23 01:34:18 2021 -0600 - Franky : Quick fix to parser file. So sorry.
Tue Feb 23 01:31:01 2021 -0600 - Franky : Fixed errors for built-in functions. Currently only works for
function(argument) calls and not object.function calls.
Mon Feb 22 23:52:10 2021 -0600 - Franky : Fixed typos.
Mon Feb 22 23:32:42 2021 -0600 - Franky : Built-in functions added. Changes not yet added to photon.ml
file yet.
Mon Feb 22 23:29:14 2021 -0600 - Franky : Added print line to input file.
Mon Feb 22 19:25:27 2021 -0500 - CalumMcCartan : added bools, not, parens
Mon Feb 22 18:18:55 2021 -0500 - Akira Higaki : added function naming
Sat Feb 20 11:01:41 2021 -0500 - CalumMcCartan : Started floats, pint, bool operators
Thu Feb 18 13:35:22 2021 -0500 - CalumMcCartan : Changed a number
Thu Feb 18 13:02:32 2021 -0500 - CalumMcCartan : cm - Create make, scanner, and parser files
Tue Feb 16 14:15:57 2021 -0500 - Calum McCartan : Initial commit

```

5 Architectural Design

5.1 Compiler Diagram

The architecture of our compiler is outlined in the diagram below, which shows the main modules of the system and their associated files as well as the intermediate representations between the components.



5.2 Scanner

Given a stream of ASCII characters from a Photon source code file (.phn), the scanner identifies tokens such as identifiers, literals, and symbols using regular expressions. Any strings or characters which do not match the syntax of any token will result in an error. In addition to removing white-space, it is at this stage that comments are removed, by removing characters found between a " symbol and a newline.

5.3 Parser

The stream of tokens identified by the scanner is passed to the parser, which begins to construct an abstract syntax tree to represent the program. The root of the AST is a 'program' which is comprised of a list of variable and function declarations, which in turn consist of statements and expressions and so on. It is at this stage that the compiler is able to reject an invalid sequence of strings which are otherwise valid tokens. To remove ambiguity from the parser, some operators are given a precedence and are either left or right associative.

5.4 Semantic Checking

The next stage is to semantically check an AST to produce an SAST by associating each part of the AST with a type. This allows any program where there are type mismatches to be rejected. When two expressions of different types are combined with a binary operator, the semantic checker decides the output type (eg. $\text{int} + \text{float} = \text{float}$). Maps of declared variables and functions are

built and checked for duplicates. It is also at this stage which any aliases are replaced by other types of SAST nodes. For example, 'ALIAS' tokens used to represent primary colors are replaced with nodes which call a function to build a Pixel structure, and attributes of structures such as 'image.width' are replaced by calls to getter functions.

5.5 Code Generation

In order to generate the next intermediate representation, the code generator reads in the SAST from the semantic checker and builds llvm instructions. One important task of the code generator is to automatically build instructions which cast variables to the required type. This occurs when a binary operator is used, an assignment is made, or when an expression is evaluated which is to be used as a function argument or return value. Another task of the generator is to build instructions which prevent overflow on the pint type, and instead clamp the values between 0 and 255. This is done by casting the two pint values to integers, performing the operation, and then using select statements to clamp the result before casting back to a pint value. The generator is also responsible for declaring the built-in functions which are linked to their C-library implementations in the next step.

5.6 Linking & the Executable

The next step is to link the llvm code with our C-library functions which are primarily used for loading and saving images, as well as transforming them with utility functions. At this point, everything is in place for our executable to be built.

5.7 Module Contribution

Our team mostly completed tasks by working on features rather than modules and so everyone had at least some involvement will all of the main components. Contributors to each file are as shown below.

File	Contributors
scanner.mll	Akira, Calum, Franky, Phu
photonparse.mly	Akira, Calum, Fanky, Phu
ast.ml	Calum, Franky
semant.ml	Akira, Calum, Franky, Phu
sast.ml	Akira, Franky, Calum
codegen.ml	Akira, Calum, Franky, Phu
utils.c	Phu
Image.c	Akira, Calum, Franky, Phu
Image.h	Akira, Calum

6 Language Evolution

6.1 Photon: At the Beginning

At the beginning of the project, we quickly settled on using a C/Java-like syntax. We wanted users to be able to use a syntax with which they were familiar.

Since Photon is an image processing language, we wanted to incorporate arrays and matrices into our language, and use those as the crux of Photon. Images were originally intended to be a two-dimensional array, or a matrix.

We also wanted to use aliases for groups of pixels, such as `_red` returning a `Pixel` struct of the corresponding (r,g,b,a) values.

We wanted to create a new primitive data type called a `pint`, which is essentially an unsigned char, and can only hold a value between 0-255. No less, no more.

Lastly, the name of our programming language was born after much heated debate amongst the team members. In the words of our system architect:

`pint` code is very not cool sounding.

Thus, we settled on Photon, because it kind of sounds like the word photo, and photos/images are the core use of our language.

6.2 Rethinking Our Design Choices

Originally, we intended to create our compiler from scratch. This would've given us the ability to have more freedom in the syntax choices Photon used. However, that proved to be significantly difficult, and as a group, we decided that instead of being strongly influenced by MicroC, Photon would be based from MicroC, as advised by our TA.

We also had to take a step back from matrices and arrays. Since the primary focus of our language was images, we decided to spend our time incorporating a C image library. Arrays exist in Photon, but they act explicitly as a non-essential component of the language, and their inclusion in the language serves more as an extra feature for the user. Furthermore, matrices do not exist in our language, as they seemed significantly more complicated and space inefficient than simply using the array pointers provided by the C image library.

We also had to switch the syntax of attributions, from `dot.value` for images, to functions, which better aligned with the rest of the built-in function suite and the functionality of the C image library.

6.3 Photon: Present-day

Photon now looks much like the original Photon that our team envisioned. Our language follows a syntax similar to one we outlined in our original LRM, and our language is almost as functional as we'd like.

A Photon source code program is clean and easy to read, and our `Image` and `Pixel` functions aim to reduce the work needed to be performed by a user.

7 Test Plan

7.1 Source and Target Language Programs

Array Element Retrieval and Length

Photon's source code:

```
1 func int main()
2 {
3     int[] a;
4     a = [0,1,9,3,5];
5
6     print(a[0]);
7     print(a[1]);
8     print(a[2]);
9
10    print(a.length);
11
12    return 0;
13 }
```

Generated LLVM:

```
1 ; ModuleID = 'Photon'
2 source_filename = "Photon"
3
4 @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
5 @fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
6 @fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
7 define i32 @main() {
8 entry:
9     %a = alloca { i32*, i32* }
10    %array_size_ptr = getelementptr inbounds { i32*, i32* },
11    { i32*, i32* }* %a, i32 0, i32 0
12    %array_size = alloca i32
13    store i32 0, i32* %array_size
14    store i32* %array_size, i32** %array_size_ptr
15    %list.array = getelementptr inbounds { i32*, i32* }, {
16    i32*, i32* }* %a, i32 0, i32 1
17    %p = alloca i32, i32 1028
18    store i32* %p, i32** %list.array
19    %new_array_ptr = alloca { i32*, i32* }
20    %array_size_ptr1 = getelementptr inbounds { i32*, i32* },
21    { i32*, i32* }* %new_array_ptr, i32 0, i32 0
22    %array_size2 = alloca i32
23    store i32 0, i32* %array_size2
24    store i32* %array_size2, i32** %array_size_ptr1
25    %list.array3 = getelementptr inbounds { i32*, i32* }, {
26    i32*, i32* }* %new_array_ptr, i32 0, i32 1
```

```

23 %p4 = alloca i32, i32 1028
24 store i32* %p4, i32** %list.array3
25 call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 0)
26 call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 1)
27 call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 9)
28 call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 3)
29 call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 5)
30 %new_array = load { i32*, i32* }, { i32*, i32* }*
    %new_array_ptr
31 store { i32*, i32* } %new_array, { i32*, i32* }* %a
32 %array_get = call i32 @array_getint({ i32*, i32* }* %a,
    i32 0)
33 %printf = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get)
34 %array_get5 = call i32 @array_getint({ i32*, i32* }* %a,
    i32 1)
35 %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get5)
36 %array_get7 = call i32 @array_getint({ i32*, i32* }* %a,
    i32 2)
37 %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get7)
38 %array_size9 = call i32 @array_sizeint({ i32*, i32* }* %a)
39 %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_size9)
40 ret i32 0
41 }
42
43 declare i32 @printf(i8*, ...)

```

Load Images

Photon's source code:

```

1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     img = load("Shapes.png");

```

```

8
9     wid = width(img);
10    ht = height(img);
11    print(wid);
12    print(ht);
13
14    save(img, "ShapesSaved.png");
15
16    return 0;
17 }

```

Generated LLVM:

```

1 ; ModuleID = 'Photon'
2 source_filename = "Photon"
3
4 %PImage = type opaque
5
6 @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
7 @fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
8 @fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
9 @str = private unnamed_addr constant [11 x i8]
10     c"Shapes.png\00"
11 @str.3 = private unnamed_addr constant [16 x i8]
12     c"ShapesSaved.png\00"
13
14 define i32 @main() {
15     entry:
16     %img = alloca %PImage*
17     %wid = alloca i32
18     %ht = alloca i32
19     %load = call %PImage* @Image_load(i8* getelementptr
20         inbounds ([11 x i8], [11 x i8]* @str, i32 0, i32 0))
21     store %PImage* %load, %PImage** %img
22     %img1 = load %PImage*, %PImage** %img
23     %width = call i32 @Image_width(%PImage* %img1)
24     store i32 %width, i32* %wid
25     %img2 = load %PImage*, %PImage** %img
26     %height = call i32 @Image_height(%PImage* %img2)
27     store i32 %height, i32* %ht
28     %wid3 = load i32, i32* %wid
29     %printf = call i32 (i8*, ...) @printf(i8* getelementptr
30         inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
31         %wid3)
32     %ht4 = load i32, i32* %ht
33     %printf5 = call i32 (i8*, ...) @printf(i8* getelementptr
34         inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
35         %ht4)
36     %img6 = load %PImage*, %PImage** %img

```

```

30     %save = call i32 @Image_save(%PImage* %img6, i8*
        getelementptr inbounds ([16 x i8], [16 x i8]* @str.3,
        i32 0, i32 0))
31     ret i32 0
32 }
33
34 declare %PImage* @Image_load(i8*)
35
36 declare i32 @Image_width(%PImage*)
37
38 declare i32 @Image_height(%PImage*)
39
40 declare i32 @printf(i8*, ...)
41
42 declare i32 @Image_save(%PImage*, i8*)

```

7.2 Test Cases

Test cases are created for each piece of functionality in the language. Once a person adds a feature to the language, it is required for them to add at least one passing test case for it. We also added failing tests to verify that the type checking works properly for the new types that we have created.

Tests were performed using both white box and black box testings. Each added feature is hand tested from the lexer to print out results in parser using our manual printing script. The code generation is also hand tested with simple programs such as hello world or print image.

We also perform integration tests to involve more complicated program that involve different data types and built-in functions, blocks and statements, control flows, etc. We tried to throw in various test cases, such as declaring built in functions with different types and number of arguments, saving images before allocating space and loading the image, etc. Each test case generated by individual programmer was then verified by the tester of the team for consistency and transparency.

7.3 Test Suites & Automation

To run all of our tests, we developed a script named `testall.sh` (see Appendix) to automate our tests. For each test, `testall.sh` compiles, runs, and compares the output with an expected output file defined by us. For image testing, we created an image directory that is copied to the code directory during testing and cleaned up after. All output images from testing are put into the `images-out` directory. These images are visually inspected by a person to verify a pass or a fail. Below is the output of the automatic test suite:

```
./testall.sh
```

test-add1...OK
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-array1...OK
test-array2...OK
test-array3...OK
test-array4...OK
test-array5...OK
test-colour-alias...OK
test-decl-order...OK
test-fib...OK
test-float1...OK
test-float2...OK
test-float3...OK
test-for1...OK
test-for2...OK
test-func1...OK
test-func2...OK
test-func3...OK
test-func4...OK
test-func5...OK
test-func6...OK
test-func7...OK
test-func8...OK
test-func9...OK
test-gcd...OK
test-gcd2...OK
test-getpixel1...OK
test-global1...OK
test-global2...OK
test-global3...OK
test-hello...OK
test-if1...OK
test-if2...OK
test-if3...OK
test-if4...OK
test-if5...OK
test-if6...OK
test-imageadd1...OK
test-imageattr...OK
test-imagecreate...OK
test-imagedestroy...OK
test-imageflip...OK
test-imagegray...OK
test-imageinvert...OK

test-imageload1...OK
test-imagepaste1...OK
test-imagepaste2...OK
test-imagesubtract1...OK
test-int-to-float...OK
test-local1...OK
test-local2...OK
test-min-max1...OK
test-min-max2...OK
test-mixed-numeric-types...OK
test-numeric-casting...OK
test-ops1...OK
test-ops2...OK
test-pint-clamp...OK
test-pint...OK
test-pixel...OK
test-printhello...OK
test-setpixel1...OK
test-sqrt1...OK
test-sqrt2...OK
test-var1...OK
test-var2...OK
test-while1...OK
test-while2...OK
fail-array1...OK
fail-array2...OK
fail-assign1...OK
fail-assign2...OK
fail-assign3...OK
fail-assign4...OK
fail-colour-alias1...OK
fail-dead1...OK
fail-dead2...OK
fail-expr1...OK
fail-expr2...OK
fail-float1...OK
fail-float2...OK
fail-for1...OK
fail-for2...OK
fail-for3...OK
fail-for4...OK
fail-for5...OK
fail-func1...OK
fail-func2...OK
fail-func3...OK
fail-func4...OK


```
fail-func5...OK
fail-func6...OK
fail-func7...OK
fail-func8...OK
fail-func9...OK
fail-getpixel1...OK
fail-global1...OK
fail-global2...OK
fail-if1...OK
fail-if2...OK
fail-if3...OK
fail-imageadd...OK
fail-imagecreate1...OK
fail-imagedestroy...OK
fail-imageload1...OK
fail-imagepaste1...OK
fail-min-max1...OK
fail-missingattr...OK
fail-nomain...OK
fail-pint1...OK
fail-print...OK
fail-printb...OK
fail-printhello1...OK
fail-return1...OK
fail-return2...OK
fail-setpixel1...OK
fail-while1...OK
fail-while2...OK
make[1]: Leaving directory '/home/Photon/code'
```

7.4 Test Roles

As the tester, Phu had the separate responsibility that we had enough tests and wrote many of the failing tests that were needed. However, since every person had to write a unit test if they were pushing a new feature, writing tests was the responsibility of the person creating that feature. See section 4.7 for more details on team member responsibilities.

8 Lessons Learned

8.1 Akira

I have never used a functional language like OCaml before, so learning a new subject while also learning a new programming language was a challenge for me. Specifically, one of the biggest challenges for me was to understand how

each file worked together in creating a programming language. However, slowly, I started to be able to connect the pieces that make a compiler come together. Another unique aspect about this project was the group dynamic. Since before this my only group project was in Art of Engineering, so to be honest I was a little worried about how well it would go. I have learned that good communication, clear deadlines, and small goals are the factors that encourage proper collaboration. Upon reflection, I am glad that I was able to take this class, not only because I learned a lot, but also because I can have this project as an achievement of progress.

Advice: The project is very intimidating at first, especially at the Hello World stage. My advice is to start and try things early even if you don't understand what is going on. Putting forward that first step is the hardest, and any progress, even if it ends up completely wrong, is good progress. Lastly, I would say to continue to have clear communication with your teammates so everyone is on the same page, even if you are struggling with something. It's better to ask for help than to submit a broken piece of code.

8.2 Calum

My biggest takeaway from this project was learning what the different stages of the compiler are, as well as their different roles and interfaces. In particular I found it useful to learn what types of errors each of the stages are responsible for detecting, as this sheds a lot of light on explaining where different errors are coming from when writing code in other languages such as Java.

As well as getting to re-introduce myself to functional languages, another important takeaway was learning the benefits of using a functional language when working on this kind of project. I found OCaml very difficult to understand in the beginning, but after some time you begin to realise the potential you have to write very concise and elegant code.

My advice would be that learning OCaml combined with trying to understand the different parts of the compiler will be extremely tedious at first, but once it starts to make sense it will suddenly get a lot easier and become quite an enjoyable project.

8.3 Franky

After programming for a few years, I think it's fair to say you understand what a programming language is, the syntax, what you can and can't do, etc., but there's little to inform you of the underneath mechanisms that transform the bits and pices of code you write into digestible machine code. Taking this class really opens your eyes, especially for someone like me who knew little of what a compiler did other than yell at you for missing a semicolon.

Going into this course, I was originally obtaining a major in political science, with a concentration in computer science. However, now that we're nearing the end, and after reading Prof. Edwards' Actual Wisdom piece he wrote for Bwog, I've given serious consideration to swapping the two. Although this was one of the more complicated and technical projects I've ever worked on, there's no describing the joy and relief you get from successfully compiling and running your code.

If I had any advice to give, as repetitive as it sounds, start early. Don't work exclusively near deadlines. Even though it'll probably work itself out near the end, I urge you to avoid the stress and eye-strain.

8.4 Phu

Throughout my undergraduate years as a Data Science major and my previous professional experience in data analytic, I have mainly programmed using languages on a high level such as Python, Java, R, Visual Basics, etc. The more I developed programs and algorithms using these languages, the more I am curious about what actually works under the hood and how computers interpret and compile them to further optimize their performance and resources usage. This urges me to take a master degree in Software Systems, and the Programming Languages and Translators class from professor Edwards clearly shed lights to my understanding of how to create a functional programming language and integrate it with a compiler.

My expectation before taking the class and before working on this project is that it's going to be really challenging and tedious, and indeed, it was actually a challenging task but really fun and informative one at the same time. I learned a new functional language, OCaml, of which syntax was really confusing at first, especially its use of recursion unlike any languages I used before, but turned out to be really useful and convenient to code your own language. I learned to create a data type and make it works for a simple arithmetic program. I also learned more complicated components, such as creating built-in functions and resolving shift-reduce conflicts. As a tester of the team projects, I also discovered more about how to test each small piece of code, and got a better insight on how dev-ops actually works.

The most important takeaway from this project was the knowledge of how compilers process a programming language and its step-by-step translation into machine language that computer can use. This would be a really helpful for designing and enhancing programs and systems in the future. I really enjoy working on this project and engaging with my team members to produce an image manipulation language that is versatile and user-friendly. My advice for future students of this class is to start early, and work through this in a trial error manner. It will not be surprising that your code will fail a lot initially, so try to break it down to smaller pieces, and things will get much clearer at the

end.

9 Future Improvements

9.1 Extra Features

There are many features that we would like to implement to expand Photon's functionality:

- Currently Photon has 3 image merging functions (add, subtract, paste). Nuke, our model language, has 30. Photon could add more merge-type functions
- Other image editing functions, such as crop, rotate, and scale.
- Be able to iterate over multiple images to edit video within the language.
- Expand the types of compatible files that Photon can use

9.2 Runtime Error Handling

Another aspect of the language that is not included is runtime error handling. For example, if the loading of an image fails because the image does not exist in the file directory, then Photon does not check it. Since situations like these are common and can lead to dangerous consequences, it is an important thing to keep in mind that this feature is missing in Photon.

10 Acknowledgements and References

A note of gratitude to McDonald's for their 2006 ad campaign in Taiwan, whose photograph of Prof. Edwards served as a wonderful image sample to use in our test suite.

More importantly, our project includes array code that we partially referenced from AP++, a Fall 2018 Project. Arrays as a data structure are non-essential, and Photon's functionality remains fully intact without them. However, arrays were included in earlier stages of development while our team was trying various ideas, and after opting out of their use for image processing, in hopes of being transparent, we decided it would feel wrong not to include them given our commit history. Our team used AP++ as a reference and guide for how to implement arrays in our project. The syntax for arrays is original to our project, but their logic and implementation, specifically in `codegen.ml`, is where the references occur. Specifically, our team referenced their project in the implementation of arrays in `codegen.ml`.

Other sources we referenced includes:
VSCoDe - Fall 2018 Project
Coral - Fall 2018 Project
Nuke - Video Editing Software
stb Image Library

11 Appendix

11.1 Code Listing

11.1.1 photon.ml

```
1 (* This file was copied from MicroC *)
2
3 (* Top-level of the Photon compiler: scan & parse the input,
4    check the resulting AST and generate an SAST from it,
5    generate LLVM IR,
6    and dump the module *)
7
8 type action = Ast | Sast | LLVM_IR | Compile
9
10 let () =
11   let action = ref Compile in
12   let set_action a () = action := a in
13   let speclist = [
14     ("-a", Arg.Unit (set_action Ast), "Print the AST");
15     ("-s", Arg.Unit (set_action Sast), "Print the SAST");
16     ("-l", Arg.Unit (set_action LLVM_IR), "Print the
17      generated LLVM IR");
18     ("-c", Arg.Unit (set_action Compile),
19      "Check and print the generated LLVM IR (default)");
19   ] in
20   let usage_msg = "usage: ./photon.native [-a|-s|-l|-c]
21     [file.phn]" in
22   let channel = ref stdin in
23   Arg.parse speclist (fun filename -> channel := open_in
24     filename) usage_msg;
25
26   let lexbuf = Lexing.from_channel !channel in
27   let ast = Photonparse.program Scanner.token lexbuf in
28   match !action with
29   | Ast -> print_string (Ast.string_of_program ast)
30   | _ -> let sast = Semant.check ast in
31     match !action with
32     | Sast -> print_string (Sast.string_of_sprogram sast)
33     | LLVM_IR -> print_string (Llvm.string_of_llmodule
34       (Codegen.translate sast))
```

```

32     | Compile -> let m = Codegen.translate sast in
33     Llvm_analysis.assert_valid_module m;
34     print_string (Llvm.string_of_llmodule m)

```

11.1.2 scanner.mll

```

1  (* Ocamllex scanner for Photon *)
2
3  { open Photonparse }
4
5  let digit = ['0' - '9']
6  let digits = digit+
7
8  rule token = parse
9    [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
10 | "#"      { comment lexbuf }          (* Comments *)
11 | '('      { LPAREN }
12 | ')'      { RPAREN }
13 | '['      { LBRACK }
14 | ']'      { RBRACK }
15 | '{'      { LBRACE }
16 | '}'      { RBRACE }
17 | ';'      { SEMI }
18 | '.'      { PERIOD }
19 | ','      { COMMA }
20 | '+'      { PLUS }
21 | '-'      { MINUS }
22 | '*'      { TIMES }
23 | '/'      { DIVIDE }
24 | '='      { ASSIGN }
25 | "=="     { EQ }
26 | "!="     { NEQ }
27 | '<'      { LT }
28 | "<="     { LEQ }
29 | ">"      { GT }
30 | ">="     { GEQ }
31 | "&&"     { AND }
32 | "||"     { OR }
33 | "!"      { NOT }
34 | "func"   { FUNC }
35 | "if"     { IF }
36 | "else"   { ELSE }
37 | "for"    { FOR }
38 | "while"  { WHILE }
39 | "return" { RETURN }
40 | "int"    { INT }
41 | "pint"   { PINT }
42 | "bool"   { BOOL }

```

```

43 | "float" { FLOAT }
44 | "void" { VOID }
45 | "string" { STRING }
46 | "true" { BLIT(true) }
47 | "false" { BLIT(false) }
48 | "arr" { ARRAY }
49 | "array_add" { ARRAY_ADD }
50 | "length" { LENGTH }
51 | "Pixel" { PIXEL }
52 | "Image" { IMAGE }
53 | ('_' ['a'-'z']* ) as str { ALIAS(str) }
54 | digits as lxm { LITERAL(int_of_string lxm) }
55 | ''' ([^ ''']* as str) ''' { STRLIT(str) }
56 | digits '.' digit* ( ['e' 'E'] ['+' '-']? digits )? as
    lxm { FLIT(lxm) }
57 | ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_' ]* as
    lxm { ID(lxm) }
58 | eof { EOF }
59 | _ as char { raise (Failure("illegal character " ^
    Char.escaped char)) }
60
61 and comment = parse
62   "\n" { token lexbuf }
63 | _ { comment lexbuf }

```

11.1.3 ast.ml

```

1  (*
2   Abstract Syntax Tree and functions for printing it
3   Based on MicroC
4
5   Authors:
6   Calum McCartan (cm4114)
7   Franky Campuzano (fc2608)
8  *)
9
10 type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq
    | Greater | Geq |
11         And | Or
12
13 type uop = Neg | Not
14
15 type typ = Int | Pint | Bool | Float | Void | String |
    Array of typ | Image | Pixel
16
17 type bind = typ * string
18
19 type expr =

```

```

20     Literal of int
21     | PLiteral of int
22     | Fliteral of string
23     | BoolLit of bool
24     | StrLiteral of string
25     | Alias of string
26     | Id of string
27     | Binop of expr * op * expr
28     | Unop of uop * expr
29     | Assign of string * expr
30     | Call of string * expr list
31     | ArrayGet of string * expr
32     | ArraySize of string
33     | ArrayLiteral of expr list
34     | Attr of string * string
35     | Noexpr
36
37 type stmt =
38     Block of stmt list
39     | Expr of expr
40     | Return of expr
41     | If of expr * stmt * stmt
42     | For of expr * expr * expr * stmt
43     | While of expr * stmt
44     | ArraySet of string * expr * expr
45     | ArrayAdd of string * expr
46
47 type func_decl = {
48     typ : typ;
49     fname : string;
50     formals : bind list;
51     locals : bind list;
52     body : stmt list;
53 }
54
55 type program = bind list * func_decl list
56
57 (* Pretty-printing functions *)
58
59 let string_of_op = function
60     Add -> "+"
61     | Sub -> "-"
62     | Mult -> "*"
63     | Div -> "/"
64     | Equal -> "=="
65     | Neq -> "!="
66     | Less -> "<"
67     | Leq -> "<="
68     | Greater -> ">"
69     | Geq -> ">="

```



```

70 | And -> "&&"
71 | Or -> "||"
72
73 let string_of_uop = function
74   Neg -> "-"
75 | Not -> "!"
76
77 let rec string_of_expr = function
78   PLiteral(l)
79 | Literal(l) -> string_of_int l
80 | Alias(l)
81 | Fliteral(l) -> l
82 | StrLiteral(l) -> "\"" ^ l ^ "\""
83 | BoolLit(true) -> "true"
84 | BoolLit(false) -> "false"
85 | Id(s) -> s
86 | Binop(e1, o, e2) ->
87   string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
88   string_of_expr e2
89 | Unop(o, e) -> string_of_uop o ^ string_of_expr e
90 | Assign(v, e) -> v ^ " = " ^ string_of_expr e
91 | Call(f, el) ->
92   f ^ "(" ^ String.concat ", " (List.map string_of_expr
93   el) ^ ")"
94 | Noexpr -> ""
95 | ArrayGet(id, e) -> "array_get " ^ id ^ ", " ^
96   (string_of_expr e)
97 | ArraySize(id) -> "array_size " ^ id
98 | ArrayLiteral(_) -> "array_literal"
99 | Attr(i, a) -> i ^ "." ^ a
100
101 let rec string_of_stmt = function
102   Block(stmts) ->
103   "{\n" ^ String.concat "\n" (List.map string_of_stmt
104   stmts) ^ "}\n"
105 | Expr(expr) -> string_of_expr expr ^ ";\n";
106 | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
107 | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^
108   ")\n" ^ string_of_stmt s
109 | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
110   string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
111 | For(e1, e2, e3, s) ->
112   "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr
113   e2 ^ " ; " ^
114   string_of_expr e3 ^ ") " ^ string_of_stmt s
115 | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
116   string_of_stmt s
117 | ArraySet(id, e1, e2) -> "array_set " ^ id ^ ", " ^
118   (string_of_expr e1) ^ ", " ^ (string_of_expr e2)

```

```

111 | ArrayAdd(id, e) -> "array_add " ^ id ^ ", " ^
    string_of_expr e
112
113 let rec string_of_typ = function
114   Int -> "int"
115   | Pint -> "pint"
116   | Bool -> "bool"
117   | Float -> "float"
118   | Void -> "void"
119   | String -> "string"
120   | Image -> "Image"
121   | Pixel -> "Pixel"
122   | Array x -> (string_of_typ x) ^ "[]"
123
124 let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^
    ";\n"
125
126 let string_of_fdecl fdecl =
127   string_of_typ fdecl.typ ^ " " ^
128   fdecl.fname ^ "(" ^ String.concat ", " (List.map snd
    fdecl.formals) ^
129   ")\n{\n" ^
130   String.concat "" (List.map string_of_vdecl fdecl.locals) ^
131   String.concat "" (List.map string_of_stmt fdecl.body) ^
132   "}\n"
133
134 let string_of_program (vars, funcs) =
135   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
136   String.concat "\n" (List.map string_of_fdecl funcs)

```

11.1.4 photonparse.mly

```

1 /* Ocaml yacc parser for Photon */
2
3 %{
4 open Ast
5 %}
6
7 %token SEMI LBRACK RBRACK LPAREN RPAREN LBRACE RBRACE COMMA
    PLUS MINUS TIMES DIVIDE ASSIGN
8 %token NOT EQ NEQ LT LEQ GT GEQ AND OR
9 %token RETURN IF ELSE FOR WHILE INT PINT BOOL FLOAT VOID
    STRING FUNC
10 %token ARRAY_ADD ARRAY_GET ARRAY_SET ARRAY_SIZE PERIOD
    LENGTH
11 %token IMAGE PIXEL
12 %token <int> LITERAL
13 %token <bool> BLIT

```

```

14 %token <string> ID FLIT STRLIT ALIAS
15 %token ARRAY
16 %token EOF
17
18 %start program
19 %type <Ast.program> program
20
21 %nonassoc NOELSE
22 %nonassoc ELSE
23 %right ASSIGN
24 %left OR
25 %left AND
26 %left EQ NEQ
27 %left LT GT LEQ GEQ
28 %left PLUS MINUS
29 %left TIMES DIVIDE
30 %right NOT
31
32 %%
33
34 program:
35     decls EOF { $1 }
36
37 decls:
38     /* nothing */ { ([], []) }
39     | decls vdecl { (($2 :: fst $1), snd $1) }
40     | decls fdecl { (fst $1, ($2 :: snd $1)) }
41
42 fdecl:
43     FUNC typ ID LPAREN formals_opt RPAREN LBRACE fbody RBRACE
44     { { typ = $2;
45       fname = $3;
46       formals = List.rev $5;
47       locals = List.rev (fst $8);
48       body = List.rev (snd $8) } }
49
50 fbody:
51     /* nothing */ { ([], []) }
52     | fbody vdecl { (($2 :: fst $1), snd $1) }
53     | fbody stmt { (fst $1, ($2 :: snd $1)) }
54
55 formals_opt:
56     /* nothing */ { [] }
57     | formal_list { $1 }
58
59 formal_list:
60     typ ID { [($1,$2)] }
61     | formal_list COMMA typ ID { ($3,$4) :: $1 }
62
63 typ:

```

```

64     INT     { Int   }
65     | PINT  { Pint  }
66     | BOOL  { Bool  }
67     | FLOAT { Float }
68     | VOID  { Void  }
69     | STRING { String }
70     | IMAGE { Image }
71     | PIXEL { Pixel }
72     | typ LBRACK RBRACK { Array($1) }
73
74 vdecl:
75     typ ID SEMI { ($1, $2) }
76
77 stmt_list:
78     /* nothing */ { [] }
79     | stmt_list stmt { $2 :: $1 }
80
81 stmt:
82     expr SEMI { Expr $1
83               }
84     | RETURN expr_opt SEMI { Return $2
85               }
86     | LBRACE stmt_list RBRACE {
87       Block(List.rev $2) }
88     | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5,
89       Block([])) }
90     | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5,
91       $7) }
92     | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
93       { For($3, $5,
94       $7, $9) }
95     | WHILE LPAREN expr RPAREN stmt { While($3, $5)
96       }
97     | ID LBRACK expr RBRACK ASSIGN expr SEMI {
98       ArraySet($1, $3, $6) }
99     | ARRAY_ADD LPAREN ID COMMA expr RPAREN SEMI {
100      ArrayAdd($3, $5) }
101
102 expr_opt:
103     /* nothing */ { Noexpr }
104     | expr { $1 }
105
106 expr:
107     LITERAL { Literal($1) }
108     | FLIT { Fliteral($1) }
109     | STRLIT { StrLiteral($1)}
110     | ALIAS { Alias($1) }
111     | BLIT { BoolLit($1) }
112     | ID { Id($1) }

```

```

105 | expr PLUS      expr { Binop($1, Add,  $3) }
106 | expr MINUS    expr { Binop($1, Sub,  $3) }
107 | expr TIMES    expr { Binop($1, Mult, $3) }
108 | expr DIVIDE   expr { Binop($1, Div,  $3) }
109 | expr EQ       expr { Binop($1, Equal, $3) }
110 | expr NEQ      expr { Binop($1, Neq,  $3) }
111 | expr LT       expr { Binop($1, Less,  $3) }
112 | expr LEQ      expr { Binop($1, Leq,   $3) }
113 | expr GT       expr { Binop($1, Greater, $3) }
114 | expr GEQ      expr { Binop($1, Geq,   $3) }
115 | expr AND      expr { Binop($1, And,   $3) }
116 | expr OR       expr { Binop($1, Or,    $3) }
117 | MINUS expr %prec NOT { Unop(Neg, $2) }
118 | NOT expr      { Unop(Not, $2) }
119 | ID ASSIGN expr { Assign($1, $3) }
120 | ID LPAREN args_opt RPAREN { Call($1, $3) }
121 | LPAREN expr RPAREN { $2 }
122 | ID LBRACK expr RBRACK { ArrayGet($1,
    $3) }
123 | LBRACK args_opt RBRACK {
    ArrayLiteral($2) }
124 | ID PERIOD LENGTH { ArraySize($1)
    }
125 | ID PERIOD ID { Attr($1, $3)
    }
126
127
128 args_opt:
129 /* nothing */ { [] }
130 | args_list { List.rev $1 }
131
132 args_list:
133 expr { [$1] }
134 | args_list COMMA expr { $3 :: $1 }

```

11.1.5 sast.ml

```

1 (*
2   Semantically-checked Abstract Syntax Tree and functions
3   for printing it
4   Based on MicroC
5
6   Authors:
7   Akira Higaki (abh2171)
8   Calum McCartan (cm4114)
9   Franky Campuzano (fc2608)
10  *)

```

```

11 open Ast
12
13 type sexpr = typ * sx
14 and sarg = typ * sexpr
15 and sx =
16     SLiteral of int
17   | SPintLit of int
18   | SFliteral of string
19   | SStrLiteral of string
20   | SBoolLit of bool
21   | SId of string
22   | SBinop of sexpr * op * sexpr
23   | SUnop of uop * sexpr
24   | SAssign of string * sexpr
25   | SCall of string * sarg list
26   | SNoexpr
27   | SArrayGet of typ * string * sexpr
28   | SArrayLiteral of typ * sexpr list
29   | SArraySize of typ * string
30
31 type sstmt =
32     SBlock of sstmt list
33   | SExpr of sexpr
34   | SReturn of sexpr
35   | SIf of sexpr * sstmt * sstmt
36   | SFor of sexpr * sexpr * sexpr * sstmt
37   | SWhile of sexpr * sstmt
38   | SArraySet of typ * string * sexpr * sexpr
39   | SArrayAdd of string * sexpr
40
41 type sfunc_decl = {
42     styp : typ;
43     sfname : string;
44     sformals : bind list;
45     slocals : bind list;
46     sbody : sstmt list;
47 }
48
49 type sprogram = bind list * sfunc_decl list
50
51 (* Pretty-printing functions *)
52
53 let rec string_of_sexpr (t, e) =
54     "(" ^ string_of_typ t ^ " : " ^ (match e with
55     | SLiteral(l) -> string_of_int l
56     | SPintLit(l) -> string_of_int l
57     | SBoolLit(true) -> "true"
58     | SBoolLit(false) -> "false"
59     | SFliteral(l) -> l
60     | SStrLiteral(l) -> l

```

```

61 | SArrayGet(_, id, e) -> "array_get " ^ id ^ ", " ^
    (string_of_sexpr e)
62 | SArraySize(_, id) -> "array_size " ^ id
63 | SArrayLiteral(_) -> "array_literal"
64 | SId(s) -> s
65 | SBinop(e1, o, e2) ->
66     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
    string_of_sexpr e2
67 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
68 | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
69 | SCall(f, el) ->
70     let el = List.map snd el in
71     f ^ "(" ^ String.concat ", " (List.map
    string_of_sexpr el) ^ ")"
72 | SNoexpr -> ""
73     ) ^ ")"
74
75 let rec string_of_sstmt = function
76     SBlock(stmts) ->
77         "{\n" ^ String.concat "" (List.map string_of_sstmt
    stmts) ^ "}\n"
78 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
79 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^
    ";\n";
80 | SIf(e, s, SBlock([])) ->
81     "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
82 | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
83     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
84 | SFor(e1, e2, e3, s) ->
85     "for (" ^ string_of_sexpr e1 ^ " ; " ^
    string_of_sexpr e2 ^ " ; " ^
86     string_of_sexpr e3 ^ ") " ^ string_of_sstmt s
87 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
    string_of_sstmt s
88 | SArraySet(_, id, e1, e2) -> "array_set " ^ id ^ ", " ^
    (string_of_sexpr e1) ^ ", " ^ (string_of_sexpr e2)
89 | SArrayAdd(id, e) -> "array_add " ^ id ^ ", " ^
    string_of_sexpr e
90
91 let string_of_sfdecl fdecl =
92     string_of_typ fdecl.styp ^ " " ^
93     fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
    fdecl.sformals) ^
94     ")\n{\n" ^
95     String.concat "" (List.map string_of_vdecl fdecl.slocals)
    ^
96     String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
97     "}\n"
98
99 let string_of_sprogram (vars, funcs) =

```

```

100 String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
101 String.concat "\n" (List.map string_of_sfdecl funcs)

```

11.1.6 semant.ml

```

1  (*
2   Semantic checking for the Photon compiler
3   Based on MicroC
4
5   Authors:
6   Franky Campuzano (fc2608)
7   Akira Higaki (abh2171)
8   Calum McCartan (cm4114)
9   Phu D Pham (pdp2121)
10  *)
11
12  open Ast
13  open Sast
14
15  module StringMap = Map.Make(String)
16
17  (* Semantic checking of the AST. Returns an SAST if
18     successful,
19     throws an exception if something is wrong.
20
21     Check each global variable, then check each function *)
22  let check (globals, functions) =
23
24    (* Verify a list of bindings has no void types or
25       duplicate names *)
26    let check_binds (kind : string) (binds : bind list) =
27      List.iter (function
28        | (Void, b) -> raise (Failure ("illegal void " ^ kind ^
29          " " ^ b))
30        | _ -> ()
31      ) binds;
32      let rec dups = function
33        | [] -> ()
34        | ((_,n1) :: (_,n2) :: _) when n1 = n2 -> raise
35          (Failure ("duplicate " ^ kind ^ " " ^ n1))
36        | _ :: t -> dups t
37      in dups (List.sort (fun (_,a) (_,b) -> compare a b)
38        binds)
39    in
40
41    (**** Check global variables ****)

```



```

38 check_binds "global" globals;
39
40 (**** Check functions ****)
41
42
43 (* Collect function declarations for built-in functions:
44    no bodies *)
45 let built_in_decls =
46   let add_bind map (name, formals', rtype) =
47     StringMap.add name {
48       (* object between brackets is func_decl object? *)
49       typ = rtype;
50       fname = name;
51       formals = formals';
52       locals = [];
53       body = []; (* empty list *)
54     } map
55   in List.fold_left add_bind StringMap.empty [
56     ("print", [(Int, "x")], Void);
57     ("printb", [(Bool, "x")], Void);
58     ("printf", [(Float, "x")], Void);
59     ("prints", [(String, "x")], Void);
60     ("min", [(Int, "x");(Int, "y")], Int);
61     ("max", [(Int, "x");(Int, "y")], Int);
62     ("sqrt", [(Float, "x")], Float);
63     ("load", [(String, "x")], Image);
64     ("save", [(Image, "img"); (String, "fname")], Void);
65     ("create", [(Int, "w"); (Int, "h"); (Pixel, "col")],
66      Image);
67     ("destroy", [(Image, "img")], Void);
68     ("flip", [(Image, "img")], Image);
69     ("to_gray", [(Image, "img")], Image);
70     ("image_paste", [(Image, "target"); (Image, "orig");
71      (Int, "x"); (Int, "y") ], Image);
72     ("image_invert", [(Image, "orig") ], Image);
73     ("image_add", [(Image, "img1"); (Image, "img2") ],
74      Image);
75     ("image_subtract", [(Image, "img1"); (Image, "img2")
76      ], Image);
77     ("get_pixel", [(Image, "img"); (Int, "x"); (Int,
78      "y")], Pixel);
79     ("set_pixel", [(Image, "img"); (Int, "x"); (Int,
80      "y"); (Pixel, "p");], Int);
81     ("width", [(Image, "img")], Int);
82     ("height", [(Image, "img")], Int);
83     ("pixel", [(Pint, "r"); (Pint, "g"); (Pint, "b");
84      (Pint, "a")], Pixel);
85     ("pixel_attr", [(Pixel, "p"); (Int, "attr")], Pint)
86   ]
87   in

```

```

79
80 (* Add function name to symbol table *)
81 let add_func map fd =
82   let built_in_err = "function " ^ fd.fname ^ " may not
      be defined"
83   and dup_err = "duplicate function " ^ fd.fname
84   and make_err er = raise (Failure er)
85   and n = fd.fname (* Name of the function *)
86   in match fd with (* No duplicate functions or
      redefinitions of built-ins *)
87     _ when StringMap.mem n built_in_decls -> make_err
      built_in_err
88     | _ when StringMap.mem n map -> make_err dup_err
89     | _ -> StringMap.add n fd map
90 in
91
92 (* Collect all function names into one symbol table *)
93 let function_decls = List.fold_left add_func
      built_in_decls functions
94 in
95
96 (* Return a function from our symbol table *)
97 let find_func s =
98   try StringMap.find s function_decls
99   with Not_found -> raise (Failure ("unrecognized
      function " ^ s))
100 in
101
102 let _ = find_func "main" in (* Ensure "main" is defined *)
103
104 let check_function func =
105   (* Make sure no formals or locals are void or
      duplicates *)
106   check_binds "formal" func.formals;
107   check_binds "local" func.locals;
108
109   (* Raise an exception if the given rvalue type cannot
      be assigned to
110   the given lvalue type *)
111   let check_assign lvaluet rvaluet err =
112     if (lvaluet = rvaluet) then lvaluet else
113     match lvaluet, rvaluet with
114       (* Types must be the same or one of these combos *)
115       | Pint, Int
116       | Int, Pint
117       | Float, Pint
118       | Float, Int -> lvaluet
119       | _ -> raise (Failure err)
120   in
121

```

```

122   (* Build local symbol table of variables for this
function *)
123   let symbols = List.fold_left (fun m (ty, name) ->
StringMap.add name ty m)
124   StringMap.empty (globals @ func.formals @ func.locals
)
125   in
126
127   (* Return a variable from our local symbol table *)
128   let type_of_identifier s =
129     try StringMap.find s symbols
130     with Not_found -> raise (Failure ("undeclared
identifier " ^ s))
131   in
132
133   let check_array_type id =
134     match (type_of_identifier id) with
135     Array t -> t
136     | t -> raise (Failure ("check array type error,
typ: " ^ string_of_typ t))
137   in
138
139   let combine_numeric_types t1 t2 =
140     if t1 = t2 then t1 else match t1, t2 with
141     (* Use the more precise type *)
142     | Int, Float | Float, Int -> Float
143     | Pint, Float | Float, Pint -> Float
144     | Pint, Int | Int, Pint -> Int
145     | _ -> raise (Failure ("cannot combine numeric
types " ^ string_of_typ t1 ^ " & " ^ string_of_typ t2))
146   in
147
148   (* Return a semantically-checked expression, i.e., with
a type *)
149   let rec expr = function
150     Literal l -> (Int, SLiteral l)
151     | PLiteral l -> (Pint, SPintLit l)
152     | Fliteral l -> (Float, SFliteral l)
153     | StrLiteral l -> (String, SStrLiteral l)
154     | Alias n ->
155     let (r, g, b, a) = (match n with
156     | "_black" -> (0, 0, 0, 255)
157     | "_white" -> (255, 255, 255, 255)
158     | "_grey" -> (128, 128, 128, 255)
159     | "_red" -> (255, 0, 0, 255)
160     | "_green" -> (0, 255, 0, 255)
161     | "_blue" -> (0, 0, 255, 255)
162     | "_cyan" -> (0, 255, 255, 255)
163     | "_magenta" -> (255, 0, 255, 255)
164     | "_yellow" -> (255, 255, 0, 255)

```

```

165         | _ -> raise (Failure ("alias " ^ n ^ " does not
166         exist"))
167         ) in expr (Call ("pixel", [PLiteral(r);
168         PLiteral(g); PLiteral(b); PLiteral(a)]))
169         | BoolLit l -> (Bool, SBoolLit l)
170         | Noexpr -> (Void, SNoexpr)
171         | Id s -> (type_of_identifier s, SId s)
172         | Attr(var, attr) ->
173         let lt = type_of_identifier var in
174         let err = var ^ " has no attribute " ^ attr in
175         (match lt with
176         | Pixel ->
177         (match attr with
178         | "r" -> expr (Call("pixel_attr", [Id(var);
179         Literal(0)]))
180         | "g" -> expr (Call("pixel_attr", [Id(var);
181         Literal(1)]))
182         | "b" -> expr (Call("pixel_attr", [Id(var);
183         Literal(2)]))
184         | "a" -> expr (Call("pixel_attr", [Id(var);
185         Literal(3)]))
186         | _ -> raise (Failure (err)))
187         | Image ->
188         (match attr with
189         | "width" -> expr (Call("width", [Id(var)]))
190         | "height" -> expr (Call("height",
191         [Id(var)]))
192         | _ -> raise (Failure (err)))
193         | _ -> raise (Failure (err)))
194         | Assign(var, e) as ex ->
195         let lt = type_of_identifier var
196         and (rt, e') = expr e in
197         let err = "illegal assignment " ^ string_of_typ
198         lt ^ " = " ^
199         string_of_typ rt ^ " in " ^ string_of_expr ex
200         in (check_assign lt rt err, SAssign(var, (rt,
201         e'))))
202         | Unop(op, e) as ex ->
203         let (t, e') = expr e in
204         let ty = match op with
205         Neg when t = Int || t = Float -> t
206         | Not when t = Bool -> Bool
207         | _ -> raise (Failure ("illegal unary operator " ^
208         string_of_uop op ^
209         string_of_typ t ^
210         " in " ^ string_of_expr
211         ex))
212         in (ty, SUnop(op, (t, e'))))
213         | Binop(e1, op, e2) as e ->
214         let (t1, e1') = expr e1

```

```

204         and (t2, e2') = expr e2 in
205         (match(t1, op, t2) with
206         (* Special cases, such as image
addition/subtraction *)
207         | (Image, Add, Image) -> expr
(Call("image_add", [e1; e2]))
208         | (Image, Sub, Image) -> expr
(Call("image_subtract", [e1; e2]))
209         | _ ->
210         let same = t1 = t2 in
211         let both_numeric =
212         ((t1 = Int) || (t1 = Pint) || (t1 = Float)) &&
213         ((t2 = Int) || (t2 = Pint) || (t2 = Float))
214         in
215         (* Determine expression type based on operator
and operand types *)
216         (* Math ops require any two numeric types,
logical ops require two bools *)
217         let ty = match op with
218         | Add | Sub | Mult | Div           when both_numeric
-> combine_numeric_types t1 t2
219         | Less | Leq | Greater | Geq      when both_numeric
-> Bool
220         | Equal | Neq                     when same ||
both_numeric -> Bool
221         | And | Or                        when same && t1 =
Bool -> Bool
222         | _ -> raise (
223         Failure ("illegal binary operator " ^
224         string_of_typ t1 ^ " " ^
string_of_op op ^ " " ^
225         string_of_typ t2 ^ " in " ^
string_of_expr e))
226         in (ty, SBinop((t1, e1'), op, (t2, e2'))))
227         | ArrayGet (var, e) ->
228         let (t, e') = expr e in
229         let ty = match t with
230         Int -> Int
231         | _ -> raise (Failure ("array_get index must
be integer, not " ^ string_of_typ t))
232         in let array_type = check_array_type var
233         in (array_type, SArrayGet(array_type, var, (ty,
e'))))
234         | ArraySize var ->
235         (Int, SArraySize(check_array_type var, var))
236         | ArrayLiteral vals ->
237         let (t', _) = expr (List.hd vals) in
238         let map_func lit = expr lit in
239         let vals' = List.map map_func vals in
240         (Array t', SArrayLiteral(t', vals'))

```

```

241 | Call(fname, args) as call ->
242   let fd = find_func fname in
243   let param_length = List.length fd.formals in
244   if List.length args != param_length then
245     raise (Failure ("expecting " ^ string_of_int
246 param_length ^
247                       " arguments in " ^
248 string_of_expr call))
249   else let check_call (ft, _) e =
250     let (et, e') = expr e in
251     let err = "illegal argument found " ^
252 string_of_typ et ^
253           " expected " ^ string_of_typ ft ^ " in " ^
254 string_of_expr e
255     in (check_assign ft et err, (et, e'))
256   in
257   let args' = List.map2 check_call fd.formals args
258   in (fd.typ, SCall(fname, args'))
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

279     | ArrayAdd (var, e) ->
280       let _ = check_array_type var in
281       SArrayAdd(var, check_match_array_type_expr var e)
282     | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt
283       b1, check_stmt b2)
284     | For(e1, e2, e3, st) ->
285       SFor(expr e1, check_bool_expr e2, expr e3, check_stmt
286       st)
287     | While(p, s) -> SWhile(check_bool_expr p, check_stmt
288       s)
289     | Return e -> let (t, e') = expr e in
290       let err = ("return gives " ^ string_of_ttyp t ^ "
291       expected " ^ string_of_ttyp func.ttyp ^ " in " ^
292       string_of_expr e) in
293       if check_assign func.ttyp t err = func.ttyp then
294         SReturn (t, e')
295       else raise (Failure err)
296
297     (* A block is correct if each statement is correct
298     and nothing
299     follows any Return statement.  Nested blocks are
300     flattened. *)
301     | Block s1 ->
302       let rec check_stmt_list = function
303         [Return _ as s] -> [check_stmt s]
304       | Return _ :: _ -> raise (Failure "nothing
305       may follow a return")
306       | Block s1 :: ss -> check_stmt_list (s1 @ ss)
307     (* Flatten blocks *)
308     | s :: ss -> check_stmt s ::
309     check_stmt_list ss
310     | [] -> []
311     in SBlock(check_stmt_list s1)
312
313   in (* body of check_function *)
314   { styp = func.ttyp;
315     sfname = func.fname;
316     sformals = func.formals;
317     slocals = func.locals;
318     sbody = match check_stmt (Block func.body) with
319     | SBlock(s1) -> s1
320     | _ -> raise (Failure ("internal error: block
321     didn't become a block?"))
322   }
323   in (globals, List.map check_function functions)

```

11.1.7 codegen.ml

```

1  (*
2   Code generation: translate takes a semantically checked
   AST and produces LLVM IR
3   Based on MicroC
4
5   Authors:
6   Akira Higaki (abh2171)
7   Calum McCartan (cm4114)
8   Franky Campuzano (fc2608)
9   Phu D Pham (pdp2121)
10  *)
11
12  module L = Llvm
13  module A = Ast
14  open Sast
15
16  module StringMap = Map.Make(String)
17
18  (* translate : Sast.program -> Llvm.module *)
19  let translate (globals, functions) =
20    let context      = L.global_context () in
21
22    (* Create the LLVM compilation module into which
23     we will generate code *)
24    let the_module = L.create_module context "Photon" in
25
26    (* Get types from the context *)
27    let i32_t      = L.i32_type    context
28    and i8_t       = L.i8_type     context
29    and i1_t       = L.i1_type     context
30    and float_t    = L.double_type context
31    and string_t   = L.pointer_type (L.i8_type context)
32    and void_t     = L.void_type   context
33    and image_t    = L.pointer_type (L.named_struct_type
   context "PImage")
34    and pixel_t    = L.i32_type    context
35    and array_t t  = L.struct_type context [| L.pointer_type
   (L.i32_type context); (L.pointer_type t) |]
36  in
37
38  (* Return the LLVM type for a Photon type *)
39  let rec ltype_of_typ = function
40    | A.Int    -> i32_t
41    | A.Pint   -> i8_t
42    | A.Bool   -> i1_t
43    | A.Float  -> float_t
44    | A.Void   -> void_t
45    | A.String -> string_t
46    | A.Image  -> image_t

```



```

47     | A.Pixel -> pixel_t
48     | A.Array t -> array_t (ltype_of_typ t)
49 in
50 let type_str t =
51     match t with
52     | A.Int -> "int"
53     | A.Pint -> "pint"
54     | A.Bool -> "bool"
55     | A.Float -> "float"
56     | A.String -> "str"
57     | _ -> raise (Failure "Invalid string map key type")
58 in
59 (* Create a map of global variables after creating each *)
60 let global_vars : L.llvalue StringMap.t =
61     let global_var m (t, n) =
62         let init = match t with
63             | A.Float -> L.const_float (ltype_of_typ t) 0.0
64             | _ -> L.const_int (ltype_of_typ t) 0
65         in StringMap.add n (L.define_global n init
66             the_module) m
67     in List.fold_left global_var StringMap.empty globals
68 in
69 (* built-in functions *)
70
71 let func_decl name =
72     let (func_t, c_name) = match name with
73     | "printf" ->
74         L.var_arg_function_type i32_t [| L.pointer_type
75             i8_t |], "printf"
76     | "min" ->
77         L.function_type i32_t [| i32_t; i32_t |], "get_min"
78     | "max" ->
79         L.function_type i32_t [| i32_t; i32_t |], "get_max"
80     | "sqrt" ->
81         L.function_type float_t [| float_t |], "get_sqrt"
82     | "load" ->
83         L.function_type image_t [| string_t |], "Image_load"
84     | "save" ->
85         L.function_type i32_t [| image_t; string_t |],
86         "Image_save"
87     | "create" ->
88         L.function_type image_t [| i32_t; i32_t; pixel_t
89             |], "Image_create"
90     | "width" ->
91         L.function_type i32_t [| image_t |], "Image_width"
92     | "height" ->
93         L.function_type i32_t [| image_t |], "Image_height"
94     | "destroy" ->
95         L.function_type i32_t [| image_t |], "Image_free"

```

```

93     | "flip" ->
94       L.function_type image_t [| image_t|], "Image_flip"
95     | "to_gray" ->
96       L.function_type image_t [| image_t|],
97       "Image_to_gray"
98     | "image_paste" ->
99       L.function_type image_t [| image_t; image_t; i32_t;
100 i32_t|], "Image_paste"
101     | "image_invert" ->
102       L.function_type image_t [| image_t|], "Image_invert"
103     | "image_add" ->
104       L.function_type image_t [| image_t; image_t|],
105       "Image_add"
106     | "image_subtract" ->
107       L.function_type image_t [| image_t; image_t|],
108       "Image_subtract"
109     | "get_pixel" ->
110       L.function_type pixel_t [| image_t; i32_t; i32_t
111 |], "get_pixel"
112     | "set_pixel" ->
113       L.function_type i32_t [| image_t; i32_t; i32_t;
114 pixel_t |], "set_pixel"
115     | "pixel" ->
116       L.function_type pixel_t [| i8_t; i8_t; i8_t; i8_t;
117 |], "pixel"
118     | "pixel_attr" ->
119       L.function_type i8_t [| pixel_t; i32_t |],
120       "pixel_attr"
121     | _ ->
122       raise (Failure "internal error: built-in func does
123 not exist ")
124     in
125     L.declare_function c_name func_t the_module
126 in
127
128 (* LLVM insists each basic block end with exactly one
129 "terminator"
130 instruction that transfers control. This function
131 runs "instr builder"
132 if the current block does not already have a
133 terminator. Used,
134 e.g., to handle the "fall off the end of the
135 function" case. *)
136
137 let add_terminal builder instr =
138   match L.block_terminator (L.insertion_block builder)
139   with
140   | Some _ -> ()
141   | None -> ignore (instr builder)
142 in

```

```

129
130 (* array functions *)
131
132 (*
133 the following array code partially referenced from AP++
134 2018 project:
135
136 http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/reports/AP++.tar.gz
137 *)
138 (* ltype array_get(array a, i32_t index) *)
139 let array_get : L.lvalue StringMap.t =
140   let array_get_ty m typ =
141     let ltype = (ltype_of_tpy typ) in
142
143     (* define the function type *)
144     let def_name = (type_str typ) in
145     let def = L.define_function ("array_get" ^ def_name)
146     (L.function_type ltype [| L.pointer_type (array_t
147 ltype); i32_t |]) the_module in
148
149     (* create array pointer *)
150     let build = L.builder_at_end context (L.entry_block
151 def) in
152     let array_ptr = L.build_alloca (L.pointer_type
153 (array_t ltype)) "array_ptr_alloc" build in
154     let _ = L.build_store (L.param def 0) array_ptr build
155 in
156
157     (* create index pointer *)
158     let index_ptr = L.build_alloca i32_t "index_alloc"
159 build in
160     let _ = L.build_store (L.param def 1) index_ptr build
161 in
162
163     (* more building and allocating *)
164     let array_load = L.build_load array_ptr "array_load"
165 build in
166     let array_ar_ptr = L.build_struct_gep array_load 1
167 "array_ar_ptr" build in
168     let array_ar_load = L.build_load array_ar_ptr
169 "array_load" build in
170
171     (* get return value *)
172     let index = L.build_load index_ptr "index_load" build
173 in
174     let array_ar_elem_ptr = L.build_gep array_ar_load [|
175 index |] "list_array_element_ptr" build in
176     let ele_val = L.build_load array_ar_elem_ptr
177 "array_ar_elem_ptr" build in

```

```

164     let _ = L.build_ret ele_val build in
165
166     StringMap.add def_name def m in
167
168 List.fold_left array_get_ty StringMap.empty [ A.Pint;
169     A.Bool; A.Int; A.Float; A.String ] in
170
171 (* void array_set(array a, i32_t index, ltype value) *)
172 let array_set : L.llvalue StringMap.t =
173     let array_set_ty m typ =
174         let ltype = (ltype_of_typ typ) in
175         let def_name = (type_str typ) in
176         let def = L.define_function ("array_set" ^ def_name)
177         (L.function_type void_t [| L.pointer_type (array_t
178         ltype); i32_t; ltype |]) the_module in
179         let build = L.builder_at_end context (L.entry_block
180         def) in
181
182         let array_ptr = L.build_alloca (L.pointer_type
183         (array_t ltype)) "array_ptr_alloc" build in
184         ignore(L.build_store (L.param def 0) array_ptr build);
185
186         let array_load = L.build_load array_ptr "array_load"
187         build in
188         let array_ar_ptr = L.build_struct_gep array_load 1
189         "array_ar_ptr" build in
190         let array_ar_load = L.build_load array_ar_ptr
191         "array_ar_load" build in
192
193         let index_element_ptr = L.build_gep array_ar_load [|
194         L.param def 1 |] "array_ar_next_ele_ptr" build in
195         let _ = L.build_store (L.param def 2)
196         index_element_ptr build in
197         let _ = L.build_ret_void build in
198         StringMap.add def_name def m in
199 List.fold_left array_set_ty StringMap.empty [ A.Pint;
200     A.Bool; A.Int; A.Float; A.String ] in
201
202 (* void array_add(array, ltype value) *)
203 let array_add_ty m typ =
204     let ltype = (ltype_of_typ typ) in
205     let def_name = (type_str typ) in
206     let def = L.define_function ("array_add" ^ def_name)
207     (L.function_type void_t [| L.pointer_type (array_t
208     ltype); ltype |]) the_module in
209     let build = L.builder_at_end context (L.entry_block
210     def) in
211     let array_ptr = L.build_alloca (L.pointer_type
212     (array_t ltype)) "array_ptr_alloc" build in

```

```

199     ignore(L.build_store (L.param def 0) array_ptr build);
200     let valPtr = L.build_alloca ltype "val_alloc" build in
201     ignore(L.build_store (L.param def 1) valPtr build);
202     let array_load = L.build_load array_ptr "array_load"
build in
203
204     let array_ar_ptr = L.build_struct_gep array_load 1
"array_ar_ptr" build in
205     let array_ar_load = L.build_load array_ar_ptr
"array_ar_load" build in
206     let array_size_ptr_ptr = L.build_struct_gep
array_load 0 "array_size_ptr_ptr" build in
207     let array_size_ptr = L.build_load array_size_ptr_ptr
"array_size_ptr" build in
208     let array_size = L.build_load array_size_ptr
"array_size" build in
209
210     let next_index = array_size in
211     let next_element_ptr = L.build_gep array_ar_load [|
next_index |] "array_ar_next_ele_ptr" build in
212     let next_size = L.build_add array_size (L.const_int
i32_t 1) "inc_size" build in
213     let _ = L.build_store next_size array_size_ptr build
in
214     let _ = L.build_store (L.build_load valPtr "val"
build) next_element_ptr build in
215     let _ = L.build_ret_void build in
216     StringMap.add def_name def m in
217     let array_add : L.llvalue StringMap.t =
218     List.fold_left array_add_ty StringMap.empty [ A.Pint;
A.Bool; A.Int; A.Float; A.String ] in
219
220 (* i32_t array_size(array a) *)
221 let array_size : L.llvalue StringMap.t =
222     let array_size_ty m typ =
223         let ltype = (ltype_of_typ typ) in
224         let def_name = (type_str typ) in
225
226         let def = L.define_function ("array_size" ^ def_name)
(L.function_type i32_t [| L.pointer_type (array_t ltype)
|]) the_module in
227         let build = L.builder_at_end context (L.entry_block
def) in
228
229         let array_ptr = L.build_alloca (L.pointer_type
(array_t ltype)) "array_ptr_alloc" build in
230         ignore(L.build_store (L.param def 0) array_ptr build);
231
232         let array_load = L.build_load array_ptr "array_load"
build in

```

```

233
234     let array_size_ptr_ptr = L.build_struct_gep
array_load 0 "array_size_ptr_ptr" build in
235     let array_size_ptr = L.build_load array_size_ptr_ptr
"array_size_ptr" build in
236     let array_size = L.build_load array_size_ptr
"array_size" build in
237     ignore(L.build_ret array_size build);
238     StringMap.add def_name def m in
239     List.fold_left array_size_ty StringMap.empty [
A.Pint; A.Bool; A.Int; A.Float; A.String ] in
240
241 (* building the array *)
242 let init_array builder array_ptr array_type =
243     (* make a size pointer and set size of the array to 0 *)
244     let sizePtrPtr = L.build_struct_gep array_ptr 0
"array_size_ptr" builder in
245     let sizePtr = L.build_alloca i32_t "array_size"
builder in
246     let _ = L.build_store (L.const_int i32_t 0) sizePtr
builder in
247     ignore(L.build_store sizePtr sizePtrPtr builder);
248     (* create the array *)
249     let array_ar_ptr = L.build_struct_gep array_ptr 1
"list.array" builder in
250     let p = L.build_array_alloca (ltype_of_typ
array_type) (L.const_int i32_t 1028) "p" builder in
251     ignore(L.build_store p array_ar_ptr builder);
252 in
253
254 (*Image Functions*)
255
256
257 (* Define each function (arguments and return type) so we
can
258 call it even before we've created its body *)
259 let function_decls : (L.llvalue * sfunc_decl) StringMap.t
=
260     let function_decl m fdecl =
261         let name = fdecl.sfname
262         and formal_types = Array.of_list (List.map (fun (t,_)
-> ltype_of_typ t) fdecl.sformals) in
263         let
264             ftype = L.function_type (ltype_of_typ fdecl.styp)
formal_types
265         in
266             StringMap.add name (L.define_function name ftype
the_module, fdecl) m
267         in
268         List.fold_left function_decl StringMap.empty functions

```

```

269   in
270
271   (* Fill in the body of the given function *)
272   let build_function_body fdecl =
273     let (the_function, _) = StringMap.find fdecl.sfname
274       function_decls in
275     let builder = L.builder_at_end context (L.entry_block
276       the_function) in
277
278     let int_format_str = L.build_global_stringptr "%d\n"
279       "fmt" builder
280     and str_format_str = L.build_global_stringptr "%s\n"
281       "fmt" builder
282     and float_format_str = L.build_global_stringptr "%g\n"
283       "fmt" builder in
284
285     (* Construct the function's "locals": formal arguments
286     and locally
287     declared variables. Allocate each on the stack,
288     initialize their
289     value, if appropriate, and remember their values in
290     the "locals" map *)
291     let local_vars =
292       let add_formal m (t, n) p =
293         L.set_value_name n p;
294         let local = L.build_alloca (ltype_of_typ t) n
295         builder in
296         ignore(
297           match t with
298             A.Array array_type -> init_array builder local
299             array_type
300             | _ -> ()
301           );
302         ignore (L.build_store p local builder);
303         StringMap.add n local m
304         (* Allocate space for any locally declared
305         variables and add the
306         * resulting registers to our map *)
307         and add_local m (t, n) =
308           let local_var = L.build_alloca (ltype_of_typ t) n
309           builder in
310           ignore(
311             match t with
312               A.Array array_type -> init_array builder
313               local_var array_type
314               | _ -> ()
315             );
316           StringMap.add n local_var m in
317       let formals = List.fold_left2 add_formal
318         StringMap.empty fdecl.sformals

```

```

305         (Array.to_list (L.params the_function))
306         in List.fold_left add_local formals fdecl.slocals
307     in
308
309 let clamp v min max =
310     if v > max then max else
311     if v < min then min else v
312 in
313
314 (* Return the value for a variable or formal argument.
315    Check local names first, then global names *)
316 let lookup n =
317     try StringMap.find n local_vars with Not_found ->
318     StringMap.find n global_vars
319 in
320
321 (* Clamp given integer between 0-255 so its ready to be
322    cast to a pint *)
323 let build_pint_clamp e builder =
324     let max = (L.const_int i32_t 255) in
325     let max_cond = L.build_icmp L.Icmp.Sgt e max
326     "compareMax" builder in
327     let e = L.build_select max_cond max e "selectMax"
328     builder in
329     let min = (L.const_int i32_t 0) in
330     let min_cond = L.build_icmp L.Icmp.Slt e min
331     "compareMin" builder in
332     L.build_select min_cond min e "selectMin" builder
333 in
334
335 (* Cast an evaluated expression 'e' from type 'rt' to
336    type 'lt' *)
337 let cast_expr e lt rt builder =
338     if lt = rt then e else
339     let llt = ltype_of_typ lt in
340     match lt, rt with
341     | A.Pint, A.Int    -> let e' = build_pint_clamp e
342     builder in L.build_trunc e' llt "pintCast" builder
343     | A.Int, A.Pint    -> L.build_zext e llt "intCast"
344     builder
345     | A.Float, A.Pint -> L.build_uitofp e llt "floatCast"
346     builder
347     | A.Float, A.Int  -> L.build_sitofp e llt "floatCast"
348     builder
349     | _ -> raise (Failure "internal error: semant should
350     have rejected an unsupported type conversion")
351 in
352
353 (* Construct code for an expression; return its value *)
354 let rec expr builder ((t, e) : sexpr) = match e with

```



```

343 | SLiteral i   -> L.const_int i32_t i
344 | SPintLit p  -> L.const_int i8_t (clamp p 0 255)
345 | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
346 | SFliteral l -> L.const_float_of_string float_t l
347 | SStrLiteral s -> L.build_global_stringptr s "str"
builder
348 | SNoexpr      -> L.const_int i32_t 0
349 | SId s         -> L.build_load (lookup s) s builder
350 | SAssign (s, (rt, e)) ->
351     let e' = expr builder (rt, e) in
352     let e' = cast_expr e' t rt builder in
353     ignore(L.build_store e' (lookup s) builder); e'
354 | SBinop ((rt1, e1), op, (rt2, e2)) ->
355     let is_pint_op = (rt1 = A.Pint && rt2 = A.Pint && t =
A.Pint) in
356     let cast_t =
357         (* If both types are pint, cast to int so we dont
overflow. *)
358         if is_pint_op then A.Int
359         (* If binop type is a bool, then cast both
expressions to float for comparision *)
360         else if t = A.Bool then
361             if rt1 = rt2 then rt1 else A.Float
362         else t
363     in
364     (* Evaluate both expressions and cast to same type
'cast_t' *)
365     let e1' = expr builder (rt1, e1)
366     and e2' = expr builder (rt2, e2) in
367     let e1' = cast_expr e1' cast_t rt1 builder
368     and e2' = cast_expr e2' cast_t rt2 builder in
369
370     let result =
371         if cast_t = A.Float then (match op with
372             | A.Add      -> L.build_fadd
373             | A.Sub      -> L.build_fsub
374             | A.Mult     -> L.build_fmud
375             | A.Div      -> L.build_fdiv
376             | A.Equal    -> L.build_fcmp L.Fcmp.Oeq
377             | A.Neq     -> L.build_fcmp L.Fcmp.One
378             | A.Less    -> L.build_fcmp L.Fcmp.Olt
379             | A.Leq     -> L.build_fcmp L.Fcmp.Ole
380             | A.Greater -> L.build_fcmp L.Fcmp.Ogt
381             | A.Geq     -> L.build_fcmp L.Fcmp.Oge
382             | A.And | A.Or -> raise (Failure "internal error:
semant should have rejected and/or on float")
383         ) e1' e2' "floatBinop" builder
384     else (match op with
385         | A.Add      -> L.build_add
386         | A.Sub      -> L.build_sub

```

```

387         | A.Mult      -> L.build_mul
388         | A.Div       -> L.build_sdiv
389         | A.And       -> L.build_and
390         | A.Or        -> L.build_or
391         | A.Equal     -> L.build_icmp L.Icmp.Eq
392         | A.Neq      -> L.build_icmp L.Icmp.Ne
393         | A.Less     -> L.build_icmp L.Icmp.Slt
394         | A.Leq      -> L.build_icmp L.Icmp.Sle
395         | A.Greater  -> L.build_icmp L.Icmp.Sgt
396         | A.Geq      -> L.build_icmp L.Icmp.Sge
397     ) e1' e2' "nonFloatBinop" builder
398
399     in
400     (* If is pint op, then cast back from int to pint *)
401     if is_pint_op then cast_expr result A.Pint A.Int
402
403     builder
404     else result
405     | SUnop(op, ((t, _) as e)) ->
406     (* Unop *)
407     let e' = expr builder e in (match op with
408     A.Neg when t = A.Float -> L.build_fneg
409     | A.Neg                -> L.build_neg
410     | A.Not                -> L.build_not) e' "tmp"
411
412     builder
413
414     (* array functions *)
415     | SArrayGet (array_type, id, e) ->
416     L.build_call (StringMap.find (type_str array_type)
417     array_get) [| (lookup id); (expr builder e) |]
418     "array_get" builder
419     | SArraySize (array_type, id) ->
420     L.build_call ((StringMap.find (type_str array_type))
421     array_size) [| (lookup id) |] "array_size" builder
422     | SArrayLiteral (array_type, literals) ->
423     let ltype = (ltype_of_tpy array_type) in
424     let new_array_ptr = L.build_alloca (array_t ltype)
425     "new_array_ptr" builder in
426     let _ = init_array builder new_array_ptr array_type in
427     let map_func literal =
428     ignore(L.build_call (StringMap.find (type_str
429     array_type) array_add) [| new_array_ptr; (expr builder
430     literal) |] "" builder);
431
432     in
433     let _ = List.rev (List.map map_func literals) in
434     L.build_load new_array_ptr "new_array" builder
435
436     | SCall (fname, f_args) ->
437     let cast_arg (lt, (rt, e)) =
438     let e' = expr builder (rt, e) in
439     cast_expr e' lt rt builder
440
441     in

```

```

429     let args = Array.of_list (List.rev (List.map
430 (cast_arg) (List.rev f_args))) in
431     let (fdef, args', result) = match fname with
432 (* Built in functions with modified arguments *)
433 | "printb"
434 | "print"    -> (func_decl "printf"), [|
int_format_str ; args.(0) |], "printf"
435 | "printf"   -> (func_decl "printf"), [|
float_format_str ; args.(0) |], "printf"
436 | "prints"   -> (func_decl "printf"), [|
str_format_str ; args.(0) |], "printf"
437 (* Built in functions with unmodified arguments *)
438 | "min" | "max" | "sqrt" | "load" | "save" |
"create" | "width"
439 | "height" | "destroy" | "flip" | "to_gray" |
"image_paste" | "image_invert" | "image_add" |
"image_subtract"
440 | "get_pixel" | "set_pixel" | "pixel" | "pixel_attr"
441 -> (func_decl fname), args, fname
442 (* User defined function *)
443 | _ ->
444     let (fdef, fdecl) = StringMap.find fname
445 function_decls in
446     let result = (match fdecl.styp with
447 | A.Void -> ""
448 | _ -> fname ^ "_result")
449     in fdef, args, result
450 in
451 L.build_call fdef args' result builder
452
453 (* Build the code for the given statement; return the
454 builder for
455 the statement's successor (i.e., the next instruction
456 will be built
457 after the one generated by this call) *)
458
459 let rec stmt builder = function
460 | SBlock sl -> List.fold_left stmt builder sl
461 | SArrayAdd (id, e) ->
462     ignore(L.build_call (StringMap.find (type_str (fst
463 e)) array_add) [| (lookup id); (expr builder e) |] ""
464 builder); builder
465 | SArraySet (array_type, id, e1, e2) ->
466     ignore(L.build_call (StringMap.find (type_str
467 array_type) array_set) [| (lookup id); (expr builder
468 e1); (expr builder e2) |] "" builder); builder
469 | SExpr e -> ignore(expr builder e); builder
470 | SReturn (t, e) ->

```

```

465     ignore(match fdecl.styp with
466             (* Special "return nothing" instr *)
467             | A.Void -> L.build_ret_void builder
468             (* Build return statement *)
469             | _ -> let e' = expr builder (t, e) in
470                   let e' = cast_expr e' fdecl.styp t builder in
471                   L.build_ret e' builder
472             ); builder
473   | SIf (predicate, then_stmt, else_stmt) ->
474       let bool_val = expr builder predicate in
475       let merge_bb = L.append_block context "merge"
476 the_function in
477       let build_br_merge = L.build_br merge_bb in (*
478 partial function *)
479
480       let then_bb = L.append_block context "then"
481 the_function in
482       add_terminal (stmt (L.builder_at_end context
483 then_bb) then_stmt)
484       build_br_merge;
485
486       let else_bb = L.append_block context "else"
487 the_function in
488       add_terminal (stmt (L.builder_at_end context
489 else_bb) else_stmt)
490       build_br_merge;
491
492       ignore(L.build_cond_br bool_val then_bb else_bb
493 builder);
494       L.builder_at_end context merge_bb
495
496   | SWhile (predicate, body) ->
497       let pred_bb = L.append_block context "while"
498 the_function in
499       ignore(L.build_br pred_bb builder);
500
501       let body_bb = L.append_block context "while_body"
502 the_function in
503       add_terminal (stmt (L.builder_at_end context body_bb)
504 body)
505       (L.build_br pred_bb);
506
507       let pred_builder = L.builder_at_end context pred_bb in
508       let bool_val = expr pred_builder predicate in
509
510       let merge_bb = L.append_block context "merge"
511 the_function in
512       ignore(L.build_cond_br bool_val body_bb merge_bb
513 pred_builder);
514       L.builder_at_end context merge_bb

```

```

503     (* Implement for loops as while loops *)
504     | SFor (e1, e2, e3, body) ->
505         stmt builder ( SBlock [SExpr e1 ; SWhile (e2, SBlock
506             [body ; SExpr e3]) ] )
507     in
508
509     (* Build the code for each statement in the function *)
510     let builder = stmt builder (SBlock fdecl.sbody) in
511
512     (* Add a return if the last block falls off the end *)
513     add_terminal builder (match fdecl.styp with
514         A.Void -> L.build_ret_void
515         | A.Float -> L.build_ret (L.const_float float_t 0.0)
516         | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
517     in
518
519     List.iter build_function_body functions;
520     the_module

```

11.1.8 Image.c

```

1  /*
2  Image library for Photon that implements the stb_image
3  library.
4  Image_load, Image_free and Image_save are written by the
5  stb_image
6  library team, the rest is written by the Photon team.
7
8  Authors:
9  Franky Campuzano (fc2608)
10 Akira Higaki (abh2171)
11 Calum McCartan (cm4114)
12 Phu D Pham (pdp2121)
13 */
14
15 #include "Image.h"
16 #include "utils.h"
17 #include <math.h>
18
19 #define STB_IMAGE_IMPLEMENTATION
20 #include "stb_image/stb_image.h"
21 #define STB_IMAGE_WRITE_IMPLEMENTATION
22 #include "stb_image/stb_image_write.h"
23
24 Image* Image_load(const char *fname) {
25     Image* img = malloc(sizeof(Image));

```

```

24     if((img->data = stbi_load(fname, &img->width,
25     &img->height, &img->channels, 4)) != NULL) {
26         img->size = img->width * img->height *
27         img->channels;
28         img->allocation_ = STB_ALLOCATED;
29     } else {
30         printf("Failed to load image %s\n", fname);
31     }
32     return img;
33 }
34
35 void Image_save(Image* img, const char* fname) {
36     if(str_ends_in(fname, ".jpg") || str_ends_in(fname,
37     ".JPG") || str_ends_in(fname, ".jpeg") ||
38     str_ends_in(fname, ".JPEG")) {
39         stbi_write_jpg(fname, img->width, img->height,
40         img->channels, img->data, 100);
41     } else if(str_ends_in(fname, ".png") ||
42     str_ends_in(fname, ".PNG")) {
43         stbi_write_png(fname, img->width, img->height,
44         img->channels, img->data, img->width * img->channels);
45     } else {
46         ON_ERROR_EXIT(false, "");
47     }
48 }
49
50 int Image_width(Image *img) {
51     return img->width;
52 }
53
54 int Image_height(Image *img) {
55     return img->height;
56 }
57
58 int get_position(int width, int channels, int x, int y) {
59     return (x + (width * y)) * (channels);
60 }
61
62 Pixel get_pixel(const Image *img, int x, int y) {
63     int pos = get_position(img->width, img-> channels, x,y);
64     unsigned char *p = img->data;
65     unsigned char pixchar = p[pos];
66
67     return pixel(p[pos], p[pos + 1], p[pos + 2], p[pos +
68     3]);
69 }
70
71 int set_pixel(Image *img, int x, int y, Pixel pix) {
72     unsigned char *p = img -> data;
73     int pos = get_position(img->width, img-> channels, x,y);

```

```

66     p[pos] = pix.r;
67     p[pos+1] = pix.g;
68     p[pos+2] = pix.b;
69     p[pos+3] = pix.a;
70
71
72     return 0;
73 }
74
75 Pixel pixel(uint8_t red, uint8_t green, uint8_t blue,
76            uint8_t alpha) {
77     Pixel p;
78     p.r = red;
79     p.g = green;
80     p.b = blue;
81     p.a = alpha;
82     return p;
83 }
84
85 uint8_t pixel_attr(Pixel p, int attr) {
86     switch(attr) {
87         case 0: return p.r;
88         case 1: return p.g;
89         case 2: return p.b;
90         case 3: return p.a;
91         default: printf("Internal error: pixel has no %d
92 attr", attr);
93     }
94     return 0;
95 }
96
97 Image* Image_create(int width, int height, Pixel col) {
98     Image* img = malloc(sizeof(Image));
99     size_t size = width * height * 4;
100    img->data = malloc(size);
101
102    if(img->data != NULL) {
103        img->width = width;
104        img->height = height;
105        img->size = size;
106        img->channels = 4;
107        img->allocation_ = SELF_ALLOCATED;
108    }
109
110    for(unsigned char *p = img->data; p != img->data +
111        img->size; p += img->channels) {
112        *p = col.r;
113        *(p + 1) = col.g;
114        *(p + 2) = col.b;
115        *(p + 3) = col.a;

```

```

113     }
114     return img;
115 }
116
117 void Image_free(Image *img) {
118     if(img->allocation_ != NO_ALLOCATION && img->data !=
119     NULL) {
120         if(img->allocation_ == STB_ALLOCATED) {
121             stbi_image_free(img->data);
122         } else {
123             free(img->data);
124         }
125         img->data = NULL;
126         img->width = 0;
127         img->height = 0;
128         img->size = 0;
129         img->allocation_ = NO_ALLOCATION;
130     }
131 }
132 Image* Image_paste( Image *gray, const Image *orig, int x,
133     int y ) {
134     //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
135     orig->channels >= 3), "The input image must have at
136     least 3 channels.");
137
138     int pos = get_position(gray->width, gray-> channels,
139     x,y);
140
141     int i = 0;
142     bool fixedpos = false;
143     for(unsigned char *p = orig->data, *pg = gray->data; p
144     != orig->data + orig->size; p += orig->channels, pg +=
145     gray->channels) {
146
147         if(!fixedpos) {
148             pg+=pos;
149             fixedpos = true;
150         }
151
152         if(i % ((orig-> width)*4) == 0 && i!= 0) {
153             pg+= (((gray-> width)*4) - (orig->width *4));
154         }
155
156         *pg = *p;
157         *(pg + 1) = *(p+1);
158         *(pg + 2) = *(p+2);
159
160         if(orig->channels == 4) {

```



```

156         *(pg + 3) = *(p + 3);
157     }
158     i+=orig-> channels;
159 }
160 return gray;
161 }
162
163
164 Image* Image_invert(Image *orig ) {
165     //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
166     orig->channels >= 3), "The input image must have at
167     least 3 channels.");
168
169     uint8_t pix;
170     int8_t maxval = 255;
171     int i = 0;
172     for(unsigned char *p = orig->data; p != orig->data +
173     orig->size; p += orig->channels) {
174         pix = *p;
175         *p = maxval - pix;
176
177         pix = *(p+1);
178         *(p + 1) = maxval - pix;
179
180         pix = *(p+2);
181         *(p + 2) = maxval - pix;
182     }
183     return orig;
184 }
185
186
187 Image* Image_add( Image *img1, Image *img2) {
188     int flag, w, h;
189     w = h = 1;
190     flag = img1->size >= img2->size ? 1 : 2;
191     int i = 0;
192     Image* gray;
193     if (flag == 1){
194         gray = Image_create(img1->width, img1->height,
195         pixel(0, 0, 0, 255));
196         for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
197         g_idx < gray->size; g_idx+= gray->channels, img1_idx +=
198         img1->channels) {
199             for (int c = 0; c < img1->channels; c++) {
200                 gray->data[g_idx + c] = img1->data[img1_idx
201                 + c];
202             }
203             if (w <= img2->width && h <= img2->height) {
204                 for (int c = 0; c < img2->channels; c++) {
205                     gray->data[g_idx + c] +=
206                     img2->data[img2_idx + c];

```

```

198         }
199         img2_idx += img2->channels;
200     }
201     if (w == gray->width) {
202         w = 1;
203         h += 1;
204     } else {
205         w += 1;
206     }
207 }
208 } else {
209     gray = Image_create(img2->width, img2->height,
210 pixel(0, 0, 0, 255));
211     for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
212 g_idx < gray->size; g_idx+= gray->channels, img2_idx +=
213 img2->channels) {
214         //printf("%d", g_idx);
215         for (int c = 0; c < img2->channels; c++) {
216             gray->data[g_idx + c] = img2->data[img2_idx
217 + c];
218         }
219         if (w <= img1->width && h <= img1->height) {
220             for (int c = 0; c < img1->channels; c++) {
221                 gray->data[g_idx + c] +=
222 img1->data[img1_idx + c];
223             }
224             img1_idx += img1->channels;
225         }
226         if (w == gray->width) {
227             w = 1;
228             h += 1;
229         } else {
230             w += 1;
231         }
232     }
233     return gray;
234 }
235
236 Image* Image_subtract( Image *img1, Image *img2) {
237     int flag, w, h;
238     w = h = 1;
239     flag = img1->size >= img2->size ? 1 : 2;
240     int i = 0;
241     Image* gray;
242     if (flag == 1){
243         gray = Image_create(img1->width, img1->height,
244 pixel(0, 0, 0, 255));
245         for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
246 g_idx < gray->size; g_idx+= gray->channels, img1_idx +=

```

```

img1->channels) {
241     for (int c = 0; c < img1->channels; c++) {
242         gray->data[g_idx + c] = img1->data[img1_idx
+ c];
243     }
244     if (w <= img2->width && h <= img2->height) {
245         for (int c = 0; c < img2->channels; c++) {
246             gray->data[g_idx + c] -=
img2->data[img2_idx + c];
247         }
248         img2_idx += img2->channels;
249     }
250     if (w == gray->width) {
251         w = 1;
252         h += 1;
253     } else {
254         w += 1;
255     }
256 }
257 } else {
258     gray = Image_create(img2->width, img2->height,
pixel(0, 0, 0, 255));
259     for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
g_idx < gray->size; g_idx+= gray->channels, img2_idx +=
img2->channels) {
260         //printf("%d", g_idx);
261         for (int c = 0; c < img2->channels; c++) {
262             gray->data[g_idx + c] = img2->data[img2_idx
+ c];
263         }
264         if (w <= img1->width && h <= img1->height) {
265             for (int c = 0; c < img1->channels; c++) {
266                 gray->data[g_idx + c] -=
img1->data[img1_idx + c];
267             }
268             img1_idx += img1->channels;
269         }
270         if (w == gray->width) {
271             w = 1;
272             h += 1;
273         } else {
274             w += 1;
275         }
276     }
277 }
278 return gray;
279 }
280
281 Image* Image_to_gray(const Image *orig) {

```

```

282     //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
orig->channels >= 3), "The input image must have at
least 3 channels.");
283     Image* gray = Image_create(orig->width, orig->height,
pixel(0, 0, 0, 255));
284     ON_ERROR_EXIT(gray->data == NULL, "Error in creating
the image");
285     uint8_t gray_p;
286
287     for(unsigned char *p = orig->data, *pg = gray->data; p
!= orig->data + orig->size; p += orig->channels, pg +=
gray->channels) {
288         gray_p = (uint8_t)((*p + *(p + 1) + *(p + 2))/3.0);
289         *pg = gray_p;
290         *(pg + 1) = gray_p;
291         *(pg + 2) = gray_p;
292         if(orig->channels == 4) {
293             *(pg + 3) = *(p + 3);
294         }
295     }
296     return gray;
297 }
298
299 Image* Image_flip(const Image *orig) {
300     //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
orig->channels >= 3), "The input image must have at
least 3 channels.");
301     int channels = 4;
302     Image* flipped = Image_create(orig->width,
orig->height, pixel(0, 0, 0, 255));
303     ON_ERROR_EXIT(flipped->data == NULL, "Error in creating
the image");
304     int index = 0;
305     int flippedIndex = 0;
306
307     for(int y = 0; y < orig->height; y++) {
308         for (int x = 0; x < orig->width; x++){
309             index = (x + (orig->width * y)) * channels;
310             flippedIndex = ((orig->width - x) +
(orig->width * y)) * channels;
311             for (int c = 0; c < channels; c++) {
312                 flipped->data[flippedIndex + c] =
orig->data[index + c];
313             }
314         }
315     }
316     return flipped;
317 }

```

11.1.9 Image.h

```
1  /*
2  Image headers for Photon that uses the stb_image library.
3  Defines the Image struct, its pointer PImage, the Pixel
4  struct, and its
5  PPixel pointer.
6  struct Image, Image_load, Image_free and Image_save are
7  written by the
8  stb_image library team. The rest is written by the Photon
9  team.
10
11  Authors:
12  Akira Higaki (abh2171)
13  Calum McCartan (cm4114)
14  */
15
16 #pragma once
17
18 #include <stdlib.h>
19 #include <stdint.h>
20 #include <stdbool.h>
21 #include <math.h>
22
23 enum allocation_type {
24     NO_ALLOCATION, SELF_ALLOCATED, STB_ALLOCATED
25 };
26
27 typedef struct {
28     int width;
29     int height;
30     int channels;
31     size_t size;
32     uint8_t *data;
33     enum allocation_type allocation_;
34 } Image;
35
36 typedef struct {
37     uint8_t r;
38     uint8_t g;
39     uint8_t b;
40     uint8_t a;
41 } Pixel;
42
43 typedef struct {
44     Image img;
45 } *PImage;
46
47 typedef struct {
```

```

45     Pixel pix;
46 } *PPixel;
47
48 Image* Image_load(const char *fname);
49 void Image_save(Image *img, const char *fname);
50 Image* Image_create(int width, int height, Pixel col);
51 void Image_free(Image *img);
52 Image* Image_to_gray(const Image *orig);
53 Image* Image_flip(const Image *orig);
54 Image* Image_add( Image *img1, Image *img2);
55 Image* Image_subtract( Image *img1, Image *img2);
56 Pixel pixel(uint8_t red, uint8_t green, uint8_t blue,
             uint8_t alpha);

```

11.1.10 utils.c

```

1  /*
2  Utility functions for Photon.
3  Based on MicroC
4
5  Authors:
6  Phu D Pham (pdp2121)
7  */
8
9  #include <stdio.h>
10 #include <math.h>
11
12 int get_max(int x, int y) {
13     if (x >= y) {
14         return x;
15     } else {
16         return y;
17     }
18 }
19
20 int get_min(int x, int y) {
21     if (x <= y){
22         return x;
23     } else {
24         return y;
25     }
26 }
27 double get_sqrt(double x) {
28     return sqrt(x);
29 }
30
31 #ifdef BUILD_TEST
32 int main()

```

```

33 {
34     char s[] = "HELLO WORLD09AZ";
35     char *c;
36     for ( c = s ; *c ; c++) printbig(*c);
37 }
38 #endif

```

11.1.11 utils.h

```

1 #pragma once
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdbool.h>
5 #include <string.h>
6
7 // Error utility macro
8 #define ON_ERROR_EXIT(cond, message) \
9 do { \
10     if((cond)) { \
11         printf("Error in function: %s at line %d\n", \
12             __func__, __LINE__); \
13         perror((message)); \
14         exit(1); \
15     } \
16 } while(0)
17
18 // Check if a string "str" ends with a substring "ends"
19 static inline bool str_ends_in(const char *str, const char
20     *ends) {
21     char *pos = strrchr(str, '.');
22     return !strcmp(pos, ends);
23 }

```

11.1.12 Makefile

```

1 # "make test" Compiles everything and runs the regression
2     tests
3
4 .PHONY : test
5 test : all testall.sh
6     ./testall.sh
7
8 retest:
9     dos2unix testall.sh
10    make clean
11    clear

```

```

11     make test
12
13 # "make all" builds the executable as well as the built-in
    library designed
14 # to test linking external code
15
16 .PHONY : all
17 all : photon.native utils.o Image.o
18
19 # "make photon.native" compiles the compiler
20 #
21 # The _tags file controls the operation of ocamlbuild,
    e.g., by including
22 # packages, enabling warnings
23 #
24 # See
    https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc
25
26 photon.native :
27     opam config exec -- \
28     ocamlbuild -use-ocamlfind photon.native
29
30 # "make clean" removes all generated files
31
32 .PHONY : clean
33 clean :
34     rm -rf testall.log ocaml11vm *.diff _build photon.native
    utils.o Image.o images-out
35     ocamlbuild -clean
36
37 utils : utils.c
38     cc -o utils -DBUILD_TEST utils.c
39
40 # Building the tarball
41
42 TARFILES = ast.ml sast.ml codegen.ml Makefile _tags
    photon.ml photonparse.mly \
43     README scanner.mll semant.ml testall.sh \
44     utils.c Image.c utils.h Image.h arcade-font.pbm font2c \
45     Dockerfile
46
47 photon.tar.gz : $(TARFILES)
48     cd .. && tar czf photon/photon.tar.gz \
49     $(TARFILES:%=photon/%)

```

11.1.13 testall.sh

```

1 #!/bin/sh

```



```

2
3 # Regression testing script for Photon
4 # Step through a list of files
5 # Compile, run, and check the output of each
   expected-to-work test
6 # Compile and check the error of each expected-to-fail test
7
8 # Copy images used in tests into root dir
9 cd images
10 {
11 for f in *.png
12 do
13     cp -v "$f" ../"$f"
14 done
15 } > /dev/null 2>&1
16 cd ..
17
18 # Path to the LLVM interpreter
19 LLI="lli"
20 #LLI="/usr/local/opt/llvm/bin/lli"
21
22 # Path to the LLVM compiler
23 LLC="llc"
24
25 # Path to the C compiler
26 CC="cc"
27
28 # Path to the photon compiler. Usually "./photon.native"
29 # Try "_build/photon.native" if ocamlbuild was unable to
   create a symbolic link.
30 Photon="./photon.native"
31 #Photon="_build/photon.native"
32
33 # Set time limit for all operations
34 ulimit -t 30
35
36 globallog=testall.log
37 rm -f $globallog
38 error=0
39 globalerror=0
40
41 keep=0
42
43 Usage() {
44     echo "Usage: testall.sh [options] [.phn files]"
45     echo "-k    Keep intermediate files"
46     echo "-h    Print this help"
47     exit 1
48 }
49

```

```

50 SignalError() {
51     if [ $error -eq 0 ] ; then
52         echo "FAILED"
53         error=1
54         fi
55         echo " $1"
56     }
57
58 # Compare <outfile> <reffile> <difffile>
59 # Compares the outfile with reffile. Differences, if any,
    written to difffile
60 Compare() {
61     generatedfiles="$generatedfiles $3"
62     echo diff -b $1 $2 ">" $3 1>&2
63     diff -b "$1" "$2" > "$3" 2>&1 || {
64     SignalError "$1 differs"
65     echo "FAILED $1 differs from $2" 1>&2
66     }
67 }
68
69 # Run <args>
70 # Report the command, run it, and report any errors
71 Run() {
72     echo $* 1>&2
73     eval $* || {
74     SignalError "$1 failed on $*"
75     return 1
76     }
77 }
78
79 # RunFail <args>
80 # Report the command, run it, and expect an error
81 RunFail() {
82     echo $* 1>&2
83     eval $* && {
84     SignalError "failed: $* did not report an error"
85     return 1
86     }
87     return 0
88 }
89
90 Check() {
91     error=0
92     basename=`echo $1 | sed 's/.*\\\/\\\/
93                                     s/.phn//'\`
94     reffile=`echo $1 | sed 's/\.phn$//'\`
95     basedir=`echo $1 | sed 's/\/\[^\\/\]*$//'\`/.'"
96
97     echo -n "$basename..."
98

```

```

99     echo 1>&2
100    echo "##### Testing $basename" 1>&2
101
102    generatedfiles=""
103
104    generatedfiles="$generatedfiles ${basename}.ll
105    ${basename}.s ${basename}.exe ${basename}.out" &&
106    Run "$Photon" "$1" ">" "${basename}.ll" &&
107    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
108    "${basename}.s" &&
109    Run "$CC" "-o" "${basename}.exe" "${basename}.s"
110    "utils.o Image.o -lm" &&
111    Run "./${basename}.exe" > "${basename}.out" &&
112    Compare ${basename}.out ${reffile}.out ${basename}.diff
113
114    # Report the status and clean up the generated files
115
116    if [ $error -eq 0 ] ; then
117    if [ $keep -eq 0 ] ; then
118        rm -f $generatedfiles
119    fi
120    echo "OK"
121    echo "##### SUCCESS" 1>&2
122    else
123    echo "##### FAILED" 1>&2
124    globalerror=$error
125    fi
126 }
127
128 CheckFail() {
129     error=0
130     basename=`echo $1 | sed 's/.*\\\/\\\/
131                s/.phn\\\/'`
132     reffile=`echo $1 | sed 's/.*.phn\\\/'`
133     basedir=`echo $1 | sed 's/\\\/[~\\\/]*$\\\/'`
134
135     echo -n "$basename..."
136
137     echo 1>&2
138     echo "##### Testing $basename" 1>&2
139
140     generatedfiles=""
141
142     generatedfiles="$generatedfiles ${basename}.err
143     ${basename}.diff" &&
144     RunFail "$Photon" "<" $1 "2>" "${basename}.err" ">>"
145     $globallog &&
146     Compare ${basename}.err ${reffile}.err ${basename}.diff
147
148     # Report the status and clean up the generated files

```

```

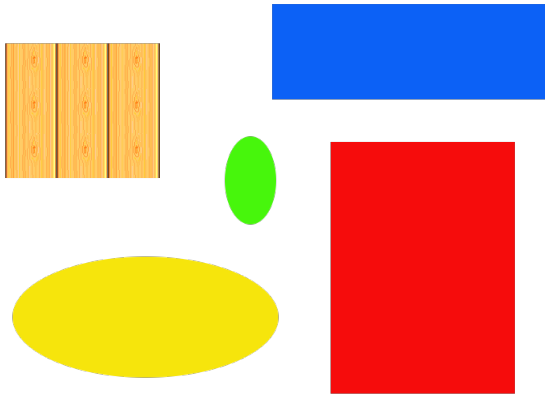
144         if [ $error -eq 0 ] ; then
145     if [ $keep -eq 0 ] ; then
146         rm -f $generatedfiles
147     fi
148     echo "OK"
149     echo "##### SUCCESS" 1>&2
150     else
151     echo "##### FAILED" 1>&2
152     globalerror=$error
153     fi
154 }
155
156 while getopts kdpsh c; do
157     case $c in
158     k) # Keep intermediate files
159         keep=1
160         ;;
161     h) # Help
162         Usage
163         ;;
164     esac
165 done
166
167 shift `expr $OPTIND - 1`
168
169 LLIFail() {
170     echo "Could not find the LLVM interpreter \"$LLI\"."
171     echo "Check your LLVM installation and/or modify the LLI
172     variable in testall.sh"
173     exit 1
174 }
175
176 which "$LLI" >> $globallog || LLIFail
177
178 if [ ! -f utils.o ]
179 then
180     echo "Could not find utils.o"
181     echo "Try \"make utils.o\""
182     exit 1
183 fi
184
185 if [ ! -f Image.o ]
186 then
187     echo "Could not find Image.o"
188     echo "Try \"make Image.o\""
189     exit 1
190 fi
191
192 if [ $# -ge 1 ]

```

```
193 then
194     files=$@
195 else
196     files="tests/test-*.phn tests/fail-*.phn"
197 fi
198
199 for file in $files
200 do
201     case $file in
202     *test-*)
203         Check $file 2>> $globallog
204         ;;
205     *fail-*)
206         CheckFail $file 2>> $globallog
207         ;;
208     *)
209         echo "unknown file type $file"
210         globalerror=1
211         ;;
212     esac
213 done
214
215 # Move output images to images-out
216 rm -f -R images-out
217 mkdir -p images-out
218 for f in *.png
219 do
220     if ! test -f images/"$f"; then
221         mv "$f" images-out/"$f"
222     fi
223 done
224 rm *.png
225
226 exit $globalerror
```

11.1.14 Sample Images

Shapes.png:



greyTest.png:



edwards.png:



edwards2.png:



11.2 Test Scripts

Our test scripts include both tests from MicroC and our custom tests for new components used in Photon.

fail-array1.phn

```
1 func int main()
2 {
3     int [] a;
4     a = [0,1,9,3,"5"];
5     print(a.length);
6
```

```
7   return 0;
8 }
```

fail-array2.phn

```
1 func int main()
2 {
3     string[] a;
4     a = [0,1,9,3];
5     print(a.length);
6
7     return 0;
8 }
```

fail-assign1.phn

```
1 func int main()
2 {
3     int i;
4     bool b;
5
6     i = 42;
7     i = 10;
8     b = true;
9     b = false;
10    i = false; # Fail: assigning a bool to an integer
11 }
```

fail-assign2.phn

```
1 func int main()
2 {
3     int i;
4     bool b;
5
6     b = 48; # Fail: assigning an integer to a bool
7 }
```

fail-assign3.phn

```
1 func void myvoid()
2 {
3     return;
4 }
```



```
5
6 func int main()
7 {
8     int i;
9
10    i = myvoid(); # Fail: assigning a void to an integer
11 }
```

fail-assign4.phn

```
1
2 func int main()
3 {
4     pint i;
5     i = 0.5; # Fail - cant assign float to pint
6 }
```

fail-colour-alias1.phn

```
1 func int main()
2 {
3     Pixel a;
4     a = _silver;
5
6     print(a.r);
7     print(a.g);
8     print(a.b);
9     print(a.a);
10
11    return 0;
12 }
```

fail-dead1.phn

```
1 func int main()
2 {
3     int i;
4
5     i = 15;
6     return i;
7     i = 32; # Error: code after a return
8 }
```

fail-dead2.phn

```
1 func int main()
2 {
3     int i;
4
5     {
6         i = 15;
7         return i;
8     }
9     i = 32; # Error: code after a return
10 }
```

fail-expr1.phn

```
1 int a;
2 bool b;
3
4 func void foo(int c, bool d)
5 {
6     int dd;
7     bool e;
8     a + c;
9     c - a;
10    a * 3;
11    c / 2;
12    d + a; # Error: bool + int
13 }
14
15 func int main()
16 {
17     return 0;
18 }
```

fail-expr2.phn

```
1 int a;
2 bool b;
3
4 func void foo(int c, bool d)
5 {
6     int d;
7     bool e;
8     b + a; # Error: bool + int
9 }
10
11 func int main()
12 {
```

```
13     return 0;
14 }
```

fail-float1.phn

```
1 func int main()
2 {
3     -3.5 && 1; # Float with AND?
4     return 0;
5 }
```

fail-float2.phn

```
1 func int main()
2 {
3     -3.5 && 2.5; # Float with AND?
4     return 0;
5 }
```

fail-for1.phn

```
1 func int main()
2 {
3     int i;
4     for ( ; true ; ) {} # OK: Forever
5
6     for (i = 0 ; i < 10 ; i = i + 1) {
7         if (i == 3) return 42;
8     }
9
10    for (j = 0; i < 10 ; i = i + 1) {} # j undefined
11
12    return 0;
13 }
```

fail-for2.phn

```
1 func int main()
2 {
3     int i;
4
5     for (i = 0; j < 10 ; i = i + 1) {} # j undefined
6
7     return 0;
```

```
8 }
```

fail-for3.phn

```
1 func int main()
2 {
3   int i;
4
5   for (i = 0; i ; i = i + 1) {} # i is an integer, not
   Boolean
6
7   return 0;
8 }
```

fail-for4.phn

```
1 func int main()
2 {
3   int i;
4
5   for (i = 0; i < 10 ; i = j + 1) {} # j undefined
6
7   return 0;
8 }
```

fail-for5.phn

```
1 func int main()
2 {
3   int i;
4
5   for (i = 0; i < 10 ; i = i + 1) {
6     foo(); # Error: no function foo
7   }
8
9   return 0;
10 }
```

fail-func1.phn

```
1 func int foo() {}
2
3 func int bar() {}
4
```

```
5 func int baz() {}
6
7 func void bar() {} # Error: duplicate function bar
8
9 func int main()
10 {
11     return 0;
12 }
```

fail-func2.phn

```
1 func int foo(int a, bool b, int c) { }
2
3 func void bar(int a, bool b, int a) {} # Error: duplicate
  formal a in bar
4
5 func int main()
6 {
7     return 0;
8 }
```

fail-func3.phn

```
1 func int foo(int a, bool b, int c) { }
2
3 func void bar(int a, void b, int c) {} # Error: illegal
  void formal b
4
5 func int main()
6 {
7     return 0;
8 }
```

fail-func4.phn

```
1 func int foo() {}
2
3 func void bar() {}
4
5 func int print() {} # Should not be able to define print
6
7 func void baz() {}
8
9 func int main()
10 {
11     return 0;
```

12 }

fail-func5.phn

```
1 func int foo() {}
2
3 func int bar() {
4     int a;
5     void b; # Error: illegal void local b
6     bool c;
7
8     return 0;
9 }
10
11 func int main()
12 {
13     return 0;
14 }
```

fail-func6.phn

```
1 func void foo(int a, bool b)
2 {
3 }
4
5 func int main()
6 {
7     foo(42, true);
8     foo(42); # Wrong number of arguments
9 }
```

fail-func7.phn

```
1 func void foo(int a, bool b)
2 {
3 }
4
5 func int main()
6 {
7     foo(42, true);
8     foo(42, true, false); # Wrong number of arguments
9 }
```

fail-func8.phn

```
1 func void foo(int a, bool b)
2 {
3 }
4
5 func void bar()
6 {
7 }
8
9 func int main()
10 {
11     foo(42, true);
12     foo(42, bar()); # int and void, not int and bool
13 }
```

fail-func9.phn

```
1 func void foo(int a, bool b)
2 {
3 }
4
5 func int main()
6 {
7     foo(42, true);
8     foo(42, 42); # Fail: int, not bool
9 }
```

fail-getpixel1.phn

```
1 func int main()
2 {
3     #whatever
4     Image img;
5
6     int index;
7     Pixel p;
8     float x;
9     int y;
10    int position;
11    int channels;
12    Pixel ps;
13
14    img = load("edwards.png");
15
16    #non integer position should return an error
17    x = 1.5;
18    y = 0;
```

```
19
20     p = get_pixel(img, x, y);
21
22     print(p.r);
23     print(p.g);
24     print(p.b);
25     print(p.a);
26
27     return 0;
28 }
```

fail-global1.phn

```
1 int c;
2 bool b;
3 void a; # global variables should not be void
4
5
6 func int main()
7 {
8     return 0;
9 }
```

fail-global2.phn

```
1 int b;
2 bool c;
3 int a;
4 int b; # Duplicate global variable
5
6 func int main()
7 {
8     return 0;
9 }
```

fail-if1.phn

```
1 func int main()
2 {
3     if (true) {}
4     if (false) {} else {}
5     if (42) {} # Error: non-bool predicate
6 }
```

fail-if2.phn

```
1 func int main()
2 {
3     if (true) {
4         foo; # Error: undeclared variable
5     }
6 }
```

fail-if3.phn

```
1 func int main()
2 {
3     if (true) {
4         42;
5     } else {
6         bar; # Error: undeclared variable
7     }
8 }
```

fail-imageadd.phn

```
1 func int main() {
2
3     Image img1;
4     img1 = load("Shapes.png");
5
6     img1 + 5;
7
8     return 0;
9 }
```

fail-imagecreate1.phn

```
1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     pint r;
8     pint g;
9
10    r = g = 0;
11    #missing b and a layers
12    img = create(800, 600, r, g);
```

```
13
14     wid = width(img);
15     ht = height(img);
16     print(wid);
17     print(ht);
18
19     save(img, "blueImgTest.png");
20
21     return 0;
22 }
```

fail-imagedestroy.phn

```
1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     img = load("Shapes.png");
8
9     wid = width(img);
10    ht = height(img);
11    print(wid);
12    print(ht);
13    #img1 undeclared
14    destroy(img1);
15
16    wid = width(img);
17    ht = height(img);
18    print(wid);
19    print(ht);
20
21    return 0;
22 }
```

fail-imageload1.phn

```
1 func int main()
2 {
3     float img;
4     int wid;
5     int ht;
6
7     #load non existed image should returns error
8     img = load("Shapes.png");
9
```

```
10     wid = width(img);
11     ht = height(img);
12     print(wid);
13     print(ht);
14
15     save(img, "ShapesSaved.png");
16
17     return 0;
18 }
```

fail-imagepaste1.phn

```
1 func int main()
2 {
3     Image imgshapes;
4     Image imgedwards;
5     Image newimg;
6     Image testing;
7     int wid;
8     int ht;
9
10    imgshapes = load("Shapes.png");
11    imgedwards = load("edwards.png");
12
13    newimg = image_paste(imgshapes, ht,0,0);
14
15    wid = width(newimg);
16    ht = height(newimg);
17    print(wid);
18    print(ht);
19
20    save(newimg, "ImgPasteTest.png");
21
22    return 0;
23 }
```

fail-min-max1.phn

```
1 func int main()
2 {
3     print(min(1,"2"));
4     return 0;
5 }
```

fail-missingattr.phn

```
1 func int main() {
2
3     Image img1;
4     img1 = load("Shapes.png");
5
6     img1.cheese;
7
8 }
```

fail-nomain.phn

fail-pint1.phn

```
1 func int main()
2 {
3     pint x;
4     x = 20.25;
5
6     print(x);
7
8     return 0;
9 }
```

fail-print.phn

```
1 # Should be illegal to redefine
2 func void print() {}
```

fail-printb.phn

```
1 # Should be illegal to redefine
2 func void printb() {}
```

fail-printlnhello1.phn

```
1 func int main()
2 {
3     #whatever
4     prints(hello world);
5     return 0;
6 }
```

fail-return1.phn

```
1 func int main()
2 {
3     return true; # Should return int
4 }
```

fail-return2.phn

```
1 func void foo()
2 {
3     if (true) return 42; # Should return void
4     else return;
5 }
6
7 func int main()
8 {
9     return 42;
10 }
```

fail-setpixel1.phn

```
1 func int main()
2 {
3     #testing set_pixel
4     Image img;
5
6     int index;
7     int position;
8     int channels;
9
10    int p1;
11    int x;
12    int y;
13
14    img = load("Shapes.png");
15
16    #for gettign a specific pixel, input the width * height
17    #coordinate you want
18    x = 0;
19    y = 0;
20    #set pixel to an int should return error
21    p1 = 30;
22
23    # Set pixel
24    set_pixel(img, x, y, p1);
25    save(img, "ShapesSavedMod.png");
```

```
25
26 # Check pixel was changed
27 img = load("ShapesSavedMod.png");
28 Pixel p2;
29 p2 = get_pixel(img, x, y);
30
31 print(p2.r);
32 print(p2.g);
33 print(p2.b);
34 print(p2.a);
35
36 return 0;
37 }
```

fail-while1.phn

```
1 func int main()
2 {
3   int i;
4
5   while (true) {
6     i = i + 1;
7   }
8
9   while (42) { # Should be boolean
10    i = i + 1;
11  }
12
13 }
```

fail-while2.phn

```
1 func int main()
2 {
3   int i;
4
5   while (true) {
6     i = i + 1;
7   }
8
9   while (true) {
10    foo(); # foo undefined
11  }
12
13 }
```

test-add1.phn

```
1 func int add(int x, int y)
2 {
3     return x + y;
4 }
5
6 func int main()
7 {
8     print( add(17, 25) );
9     return 0;
10 }
```

test-arith1.phn

```
1 func int main()
2 {
3     print(39 + 3);
4     return 0;
5 }
```

test-arith2.phn

```
1 func int main()
2 {
3     print(1 + 2 * 3 + 4);
4     return 0;
5 }
```

test-arith3.phn

```
1 func int foo(int a)
2 {
3     return a;
4 }
5
6 func int main()
7 {
8     int a;
9     a = 42;
10    a = a + 5;
11    print(a);
12    return 0;
13 }
```

test-array1.phn

```
1 func int main()
2 {
3     int [] a;
4     a = [0,1,2];
5
6     print(a[0]);
7     print(a[1]);
8     print(a[2]);
9
10    return 0;
11 }
```

test-array2.phn

```
1 func int main()
2 {
3     string [] a;
4     a = ["dog", "cat", "fish"];
5
6     prints(a[0]);
7     prints(a[1]);
8     prints(a[2]);
9
10    return 0;
11 }
```

test-array3.phn

```
1 func int main()
2 {
3     float [] a;
4     a = [1.1,2.2,3.3];
5
6     printf(a[0]);
7     printf(a[1]);
8     printf(a[2]);
9
10    return 0;
11 }
```

test-array4.phn

```
1 func int main()
2 {
```



```
3     int [] a;
4     a = [0,1,9,3,5];
5
6     print(a[0]);
7     print(a[1]);
8     print(a[2]);
9
10    print(a.length);
11
12    return 0;
13 }
```

test-array5.phn

```
1 func int main()
2 {
3     int [] a;
4     int i;
5
6     a = [0,1,9,3,6];
7
8     print(a[0]);
9     print(a[1]);
10    print(a[2]);
11
12    print(a.length);
13
14    a[0] = 21;
15    print(a[0]);
16
17
18    for (i = 0 ; i < a.length; i = i + 1) {
19        print(a[i]);
20    }
21
22    print(a.length);
23
24    return 0;
25 }
```

test-colour-alias.phn

```
1 func int main()
2 {
3     Pixel a;
4     a = _red;
5
```

```
6     print(a.r);
7     print(a.g);
8     print(a.b);
9     print(a.a);
10
11     return 0;
12 }
```

test-decl-order.phn

```
1 func int main()
2 {
3     int x;
4     x = 12;
5     print(x);
6
7     y = 5;
8     int y; # vdecls are hoisted
9
10    int z;
11    z = x + y;
12    print(z);
13
14    return 0;
15 }
```

test-fib.phn

```
1 func int fib(int x)
2 {
3     if (x < 2) return 1;
4     return fib(x-1) + fib(x-2);
5 }
6
7 func int main()
8 {
9     print(fib(0));
10    print(fib(1));
11    print(fib(2));
12    print(fib(3));
13    print(fib(4));
14    print(fib(5));
15    return 0;
16 }
```

test-float1.phn

```
1 func int main()
2 {
3     float a;
4     a = 3.14159267;
5     printf(a);
6     return 0;
7 }
```

test-float2.phn

```
1 func int main()
2 {
3     float a;
4     float b;
5     float c;
6     a = 3.14159267;
7     b = -2.71828;
8     c = a + b;
9     printf(c);
10    return 0;
11 }
```

test-float3.phn

```
1 func void testfloat(float a, float b)
2 {
3     printf(a + b);
4     printf(a - b);
5     printf(a * b);
6     printf(a / b);
7     printb(a == b);
8     printb(a == a);
9     printb(a != b);
10    printb(a != a);
11    printb(a > b);
12    printb(a >= b);
13    printb(a < b);
14    printb(a <= b);
15 }
16
17 func int main()
18 {
19     float c;
20     float d;
21 }
```

```
22     c = 42.0;
23     d = 3.14159;
24
25     testfloat(c, d);
26
27     testfloat(d, d);
28
29     return 0;
30 }
```

test-for1.phn

```
1 func int main()
2 {
3     int i;
4     for (i = 0 ; i < 5 ; i = i + 1) {
5         print(i);
6     }
7     print(42);
8     return 0;
9 }
```

test-for2.phn

```
1 func int main()
2 {
3     int i;
4     i = 0;
5     for ( ; i < 5; ) {
6         print(i);
7         i = i + 1;
8     }
9     print(42);
10    return 0;
11 }
```

test-funcl.phn

```
1 func int add(int a, int b)
2 {
3     return a + b;
4 }
5
6 func int main()
7 {
8     int a;
```

```
9   a = add(39, 3);
10  print(a);
11  return 0;
12 }
```

test-func2.phn

```
1 # Bug noticed by Pin-Chin Huang
2
3 func int fun(int x, int y)
4 {
5     return 0;
6 }
7
8 func int main()
9 {
10     int i;
11     i = 1;
12
13     fun(i = 2, i = i+1);
14
15     print(i);
16     return 0;
17 }
```

test-func3.phn

```
1 func void printem(int a, int b, int c, int d)
2 {
3     print(a);
4     print(b);
5     print(c);
6     print(d);
7 }
8
9 func int main()
10 {
11     printem(42,17,192,8);
12     return 0;
13 }
```

test-func4.phn

```
1 func int add(int a, int b)
2 {
3     int c;
```

```
4     c = a + b;
5     return c;
6 }
7
8 func int main()
9 {
10    int d;
11    d = add(52, 10);
12    print(d);
13    return 0;
14 }
```

test-func5.phn

```
1 func int foo(int a)
2 {
3     return a;
4 }
5
6 func int main()
7 {
8     return 0;
9 }
```

test-func6.phn

```
1 func void foo() {}
2
3 func int bar(int a, bool b, int c) { return a + c; }
4
5 func int main()
6 {
7     print(bar(17, false, 25));
8     return 0;
9 }
```

test-func7.phn

```
1 int a;
2
3 func void foo(int c)
4 {
5     a = c + 42;
6 }
7
8 func int main()
```

```
9 {
10     foo(73);
11     print(a);
12     return 0;
13 }
```

test-func8.phn

```
1 func void foo(int a)
2 {
3     print(a + 3);
4 }
5
6 func int main()
7 {
8     foo(40);
9     return 0;
10 }
```

test-func9.phn

```
1 func void foo(int a)
2 {
3     print(a + 3);
4     return;
5 }
6
7 func int main()
8 {
9     foo(40);
10    return 0;
11 }
```

test-gcd.phn

```
1 func int gcd(int a, int b) {
2     while (a != b) {
3         if (a > b) a = a - b;
4         else b = b - a;
5     }
6     return a;
7 }
8
9 func int main()
10 {
11     print(gcd(2,14));
```

```
12     print(gcd(3,15));
13     print(gcd(99,121));
14     return 0;
15 }
```

test-gcd2.phn

```
1 func int gcd(int a, int b) {
2     while (a != b)
3         if (a > b) a = a - b;
4         else b = b - a;
5     return a;
6 }
7
8 func int main()
9 {
10     print(gcd(14,21));
11     print(gcd(8,36));
12     print(gcd(99,121));
13     return 0;
14 }
```

test-getpixel1.phn

```
1 func int main()
2 {
3     #whatever
4     Image img;
5
6     int index;
7     Pixel p;
8     int x;
9     int y;
10    int position;
11    int channels;
12    Pixel ps;
13
14    img = load("edwards.png");
15
16    #for getting a specific pixel, input what you want
17    x = 1;
18    y = 0;
19
20    p = get_pixel(img, x, y);
21
22    print(p.r);
23    print(p.g);
```



```
24     print(p.b);
25     print(p.a);
26
27     return 0;
28 }
```

test-global1.phn

```
1  int a;
2  int b;
3
4  func void printa()
5  {
6      print(a);
7  }
8
9  func void printbb()
10 {
11     print(b);
12 }
13
14 func void incab()
15 {
16     a = a + 1;
17     b = b + 1;
18 }
19
20 func int main()
21 {
22     a = 42;
23     b = 21;
24     printa();
25     printbb();
26     incab();
27     printa();
28     printbb();
29     return 0;
30 }
```

test-global2.phn

```
1  bool i;
2
3  func int main()
4  {
5      int i; # Should hide the global i
6  }
```

```
7   i = 42;
8   print(i + i);
9   return 0;
10 }
```

test-global3.phn

```
1  int i;
2  bool b;
3  int j;
4
5  func int main()
6  {
7    i = 42;
8    j = 10;
9    print(i + j);
10   return 0;
11 }
```

test-hello.phn

```
1  func int main()
2  {
3    print(42);
4    print(71);
5    print(1);
6    return 0;
7  }
```

test-if1.phn

```
1  func int main()
2  {
3    if (true) print(42);
4    print(17);
5    return 0;
6  }
```

test-if2.phn

```
1  func int main()
2  {
3    if (true) print(42); else print(8);
4    print(17);
```

```
5     return 0;
6 }
```

test-if3.phn

```
1 func int main()
2 {
3     if (false) print(42);
4     print(17);
5     return 0;
6 }
```

test-if4.phn

```
1 func int main()
2 {
3     if (false) print(42); else print(8);
4     print(17);
5     return 0;
6 }
```

test-if5.phn

```
1 func int cond(bool b)
2 {
3     int x;
4     if (b)
5         x = 42;
6     else
7         x = 17;
8     return x;
9 }
10
11 func int main()
12 {
13     print(cond(true));
14     print(cond(false));
15     return 0;
16 }
```

test-if6.phn

```
1 func int cond(bool b)
2 {
```

```

3   int x;
4   x = 10;
5   if (b)
6       if (x == 10)
7           x = 42;
8   else
9       x = 17;
10  return x;
11 }
12
13 func int main()
14 {
15     print(cond(true));
16     print(cond(false));
17     return 0;
18 }

```

test-imageadd1.phn

```

1 func int main()
2 {
3     Image imgshapes;
4     Image imgedwards;
5     Image newimg;
6     int wid;
7     int ht;
8
9     imgshapes = load("Shapes.png");
10    imgedwards = load("edwards.png");
11
12    newimg = image_add(imgshapes, imgedwards);
13
14    wid = width(newimg);
15    ht = height(newimg);
16    print(wid);
17    print(ht);
18
19    save(newimg, "ImgAddTest.png");
20
21
22    # Using operators
23    Image newimg2;
24    newimg2 = imgshapes + imgedwards;
25    save(newimg2, "ImgAddTest2.png");
26    print(newimg2.width);
27    print(newimg2.height);
28
29    return 0;

```

30 }

test-imageattr.phn

```
1 func int main() {
2     Image img;
3     img = load("Shapes.png");
4
5     print(img.width);
6     print(img.height);
7
8     return 0;
9 }
```

test-imagecreate.phn

```
1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     img = create(800, 600, _blue);
8
9     wid = width(img);
10    ht = height(img);
11    print(wid);
12    print(ht);
13
14    save(img, "blueImgTest.png");
15
16    return 0;
17 }
```

test-imagedestroy.phn

```
1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     img = load("Shapes.png");
8
9     wid = width(img);
10    ht = height(img);
```

```
11     print(wid);
12     print(ht);
13
14     destroy(img);
15
16     wid = width(img);
17     ht = height(img);
18     print(wid);
19     print(ht);
20
21     return 0;
22 }
```

test-imageflip.phn

```
1 func int main()
2 {
3     Image img;
4     Image flippedimg;
5     int wid;
6     int ht;
7
8     img = load("Shapes.png");
9     flippedimg = flip(img);
10
11     wid = width(flippedimg);
12     ht = height(flippedimg);
13     print(wid);
14     print(ht);
15
16     save(flippedimg, "flipImgTest.png");
17
18     return 0;
19 }
```

test-imagegray.phn

```
1 func int main()
2 {
3     Image img;
4     Image grayimg;
5     int wid;
6     int ht;
7
8     img = load("Shapes.png");
9     grayimg = to_gray(img);
10
```

```
11     wid = width(grayimg);
12     ht = height(grayimg);
13     print(wid);
14     print(ht);
15
16     save(grayimg, "grayImgTest.png");
17
18     return 0;
19 }
```

test-imageinvert.phn

```
1 func int main()
2 {
3     Image imgedwards;
4     Image newimg;
5     int wid;
6     int ht;
7
8     imgedwards = load("edwards.png");
9
10    newimg = image_invert(imgedwards);
11
12    wid = width(newimg);
13    ht = height(newimg);
14    print(wid);
15    print(ht);
16
17    save(newimg, "ImgInvertTest.png");
18
19    return 0;
20 }
```

test-imageload1.phn

```
1 func int main()
2 {
3     Image img;
4     int wid;
5     int ht;
6
7     img = load("Shapes.png");
8
9     wid = width(img);
10    ht = height(img);
11    print(wid);
12    print(ht);
```

```
13
14     save(img, "ShapesSaved.png");
15
16     return 0;
17 }
```

test-imagepaste1.phn

```
1 func int main()
2 {
3     Image imgshapes;
4     Image imgedwards;
5     Image newimg;
6     int wid;
7     int ht;
8     int x;
9     int y;
10
11
12
13     imgshapes = load("Shapes.png");
14     imgedwards = load("edwards.png");
15
16
17     x = y = 0;
18     newimg = image_paste(imgshapes, imgedwards, 0,0);
19
20     wid = width(newimg);
21     ht = height(newimg);
22     print(wid);
23     print(ht);
24
25     save(newimg, "ImgPasteTest.png");
26
27     return 0;
28 }
```

test-imagepaste2.phn

```
1 func int main()
2 {
3     Image imgshapes;
4     Image imgedwards;
5     Image newimg;
6     int wid;
7     int ht;
8     int x;
```



```

9     int y;
10
11
12
13     imgshapes = load("Shapes.png");
14     imgedwards = load("edwards.png");
15
16
17     x = 300;
18     y = 300;
19     newimg = image_paste(imgshapes, imgedwards, x,y);
20
21     wid = width(newimg);
22     ht = height(newimg);
23     print(wid);
24     print(ht);
25
26     save(newimg, "ImgPasteTest2.png");
27
28     return 0;
29 }

```

test-imagesubtract1.phn

```

1 func int main()
2 {
3     Image imgshapes;
4     Image imgedwards;
5     Image newimg;
6     int wid;
7     int ht;
8
9     imgshapes = load("Shapes.png");
10    imgedwards = load("edwards.png");
11
12    newimg = image_subtract(imgshapes, imgedwards);
13
14    wid = width(newimg);
15    ht = height(newimg);
16    print(wid);
17    print(ht);
18
19    save(newimg, "ImgSubtractTest.png");
20
21
22    # Using operators
23    Image newimg2;
24    newimg2 = imgshapes - imgedwards;

```

```
25     save(newimg2, "ImgSubtractTest2.png");
26     print(newimg2.width);
27     print(newimg2.height);
28
29     return 0;
30 }
```

test-int-to-float.phn

```
1
2 func int main()
3 {
4     int i;
5     float f;
6     i = 5;
7     f = 1.7;
8     f = 1.2 + 5;
9     printf(f);
10    printf(6.4 / 2);
11    printf(2 * 2.2);
12    printf(2 + 2.2);
13    printb(2 > 2.2);
14    printb(3.14 != 3);
15 }
```

test-local1.phn

```
1 func void foo(bool i)
2 {
3     int i; # Should hide the formal i
4
5     i = 42;
6     print(i + i);
7 }
8
9 func int main()
10 {
11     foo(true);
12     return 0;
13 }
```

test-local2.phn

```
1 func int foo(int a, bool b)
2 {
3     int c;
```

```
4     bool d;
5
6     c = a;
7
8     return c + 10;
9 }
10
11 func int main() {
12     print(foo(37, false));
13     return 0;
14 }
```

test-min-max1.phn

```
1 func int main()
2 {
3     print(min(1,2));
4     print(max(1,2));
5     return 0;
6 }
```

test-min-max2.phn

```
1 func int main()
2 {
3     int a;
4     int b;
5     a = 1;
6     b = 2;
7     print (min(a,b));
8     a = 3;
9     print (max(a,b));
10    return 0;
11 }
```

test-mixed-numeric-types.phn

```
1 func int main()
2 {
3
4     int i;
5     pint p;
6     float f;
7
8     p = 10;
9     f = 0.5;
```

```

10     i = 4;
11
12     printf((p * 2) / i + f);
13     print(p + i);
14     printb(p > i);
15
16
17     return 0;
18 }

```

test-numeric-casting.phn

```

1 func int main()
2 {
3     int i;
4     pint p;
5     float f;
6
7     p = 3;
8     i = returnPintAsInt();
9     f = 9;
10
11     print(p);
12     print(i);
13     printf(f);
14
15     p = i;
16     i = p;
17     f = i + 1;
18
19     print(p);
20     print(i);
21     printf(f);
22
23     f = p;
24     printf(f);
25     useIntForFloatArg(i);
26
27     return 0;
28 }
29
30 func int returnPintAsInt() {
31     pint p;
32     p = 6;
33     return p;
34 }
35
36 func void useIntForFloatArg(float f) {

```

37 }

test-ops1.phn

```
1 func int main()
2 {
3     print(1 + 2);
4     print(1 - 2);
5     print(1 * 2);
6     print(100 / 2);
7     print(99);
8     printb(1 == 2);
9     printb(1 == 1);
10    print(99);
11    printb(1 != 2);
12    printb(1 != 1);
13    print(99);
14    printb(1 < 2);
15    printb(2 < 1);
16    print(99);
17    printb(1 <= 2);
18    printb(1 <= 1);
19    printb(2 <= 1);
20    print(99);
21    printb(1 > 2);
22    printb(2 > 1);
23    print(99);
24    printb(1 >= 2);
25    printb(1 >= 1);
26    printb(2 >= 1);
27    return 0;
28 }
```

test-ops2.phn

```
1 func int main()
2 {
3     printb(true);
4     printb(false);
5     printb(true && true);
6     printb(true && false);
7     printb(false && true);
8     printb(false && false);
9     printb(true || true);
10    printb(true || false);
11    printb(false || true);
12    printb(false || false);
```

```
13     printb(!false);
14     printb(!true);
15     print(-10);
16     print(--42);
17 }
```

test-pint-clamp.phn

```
1 func int main()
2 {
3     pint x;
4     int i;
5
6     x = 42;
7     print(x);
8     x = 255;
9     print(x);
10    x = 256;
11    print(x);
12    x = 1000;
13    print(x);
14    x = 0;
15    print(x);
16    x = -1;
17    print(x);
18
19    i = 10;
20    x = 1 + 1;
21    print(x);
22
23    pint x2;
24    x = 200;
25    x2 = 200;
26    print(x + x2);
27
28    x = getBigInt();
29    print(x);
30
31    x = x2;
32    printb(x == x2);
33
34    printPint(66);
35    printPint(-12);
36
37    return 0;
38 }
39
40 func int getBigInt() {
```

```
41     int z;  
42     z = 500;  
43     return z;  
44 }  
45  
46 func int printPint(pint p) {  
47     print(p);  
48     return p;  
49 }
```

test-pint.phn

```
1 func int main()  
2 {  
3     pint x;  
4     x = 5;  
5     print(x);  
6  
7     print(pintFunc(x));  
8  
9     return 0;  
10 }  
11  
12 func pint pintFunc(int x)  
13 {  
14     pint y;  
15     y = 3;  
16  
17     return y;  
18 }
```

test-pixel.phn

```
1 func int main()  
2 {  
3     Pixel p;  
4  
5     p = pixel(255, 128, 0, 255);  
6  
7     print(p.r);  
8     print(p.g);  
9     print(p.b);  
10    print(p.a);  
11  
12    return 0;  
13 }
```

test-printhello.phn

```
1 func int main()
2 {
3     #whatever
4     prints("hello world");
5     return 0;
6 }
```

test-setpixel1.phn

```
1 func int main()
2 {
3     #testing set_pixel
4     Image img;
5
6     int index;
7     int position;
8     int channels;
9
10    Pixel p1;
11    int x;
12    int y;
13
14    img = load("Shapes.png");
15
16    #for getting a specific pixel, input the width * height
17    #coordinate you want
18    x = 0;
19    y = 0;
20    p1 = pixel(255, 0, 0, 255);
21
22    # Set pixel
23    set_pixel(img, x, y, p1);
24    save(img, "ShapesSavedMod.png");
25
26    # Check pixel was changed
27    img = load("ShapesSavedMod.png");
28    Pixel p2;
29    p2 = get_pixel(img, x, y);
30
31    print(p2.r);
32    print(p2.g);
33    print(p2.b);
34    print(p2.a);
35
36    return 0;
37 }
```

test-sqrt1.phn

```
1 func int main()
2 {
3     printf(sqrt(4));
4     printf(sqrt(2.25));
5     printf(sqrt(-1));
6     return 0;
7 }
```

test-sqrt2.phn

```
1 func int main()
2 {
3     float a;
4     a = 9.0;
5     float b;
6     b = 0.25;
7     printf(sqrt(a));
8     printf(sqrt(b));
9 }
```

test-var1.phn

```
1 func int main()
2 {
3     int a;
4     a = 42;
5     print(a);
6     return 0;
7 }
```

test-var2.phn

```
1 int a;
2
3 func void foo(int c)
4 {
5     a = c + 42;
6 }
7
8 func int main()
9 {
10    foo(73);
11    print(a);
12    return 0;
```

13 }

test-while1.phn

```
1 func int main()
2 {
3     int i;
4     i = 5;
5     while (i > 0) {
6         print(i);
7         i = i - 1;
8     }
9     print(42);
10    return 0;
11 }
```

test-while2.phn

```
1 func int foo(int a)
2 {
3     int j;
4     j = 0;
5     while (a > 0) {
6         j = j + 2;
7         a = a - 1;
8     }
9     return j;
10 }
11
12 func int main()
13 {
14     print(foo(7));
15     return 0;
16 }
```
