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
```

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

This should give the output:

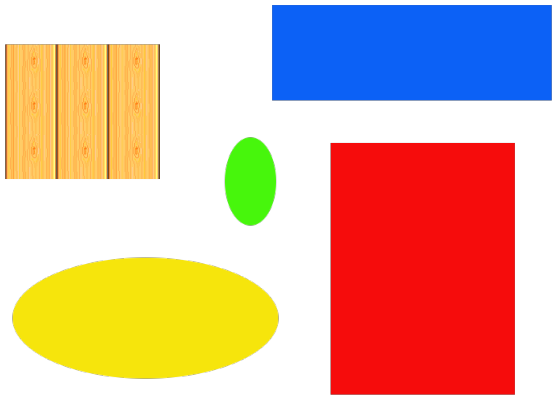
```
./test-hello.exe
42
71
1
```

2.2 Simple Image Example

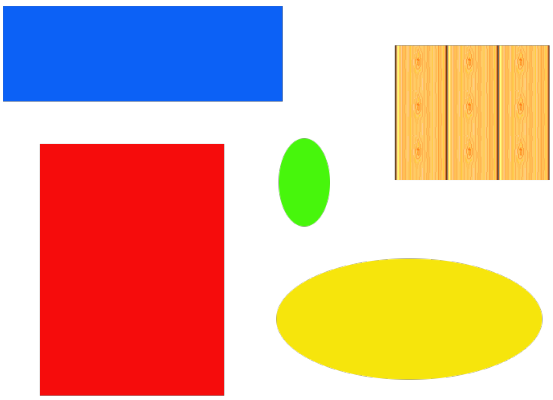
Here is a very simple example of how to edit an image using a built in function.

```
func int main()
{
  Image img;
  Image flippedimg;

  img = load("Shapes.png");
  flippedimg = flip(img);
  save(flippedimg, "flipImgTest.png");

  return 0;
}
```

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 perfomed 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
func	Function declaration
return	Followed by a value that is returned to the caller
if	Beginning of conditional statement
else	Conditional statement
for	Iterative statement
while	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.

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

For example...

```
int x;
x = 42;
int y;
bool myBool = true;
y = x + 10;
x = 3;
string greeting;
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.

```
x = 42;
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.

```
int GoodName123;
float other321_name;
```

3.7.4 String literals

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

```
string x;
x = "hello `friend'";
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.

```
Pixel myPurplePixel;
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.

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

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

```
x = myPixel.r;
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.

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

3.9.2 Width and height

The width and height attribute is accessible.

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

3.9.3 Accessing and setting a pixel

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

```
favouritePixel = get_pixel(img, 12, 18);
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:

```
myImg = load("myImage.png");
flip(myImg);
to_grey(myImg);
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:

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

Images are added with the following equation:

```
newImg.pixels[x][y].red =
img1.pixels[x][y].red * img1.pixel[x][y].alpha / 255 +
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:

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

Images are subtracted with the following equation:

```
newImg.pixel[x][y].red =
img1.pixel[x][y].red * img1.pixel[x][y].alpha / 255 -
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:

```
Image myImg;
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:

```
Image myImg;
```

```
myImg = load(filePath);
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.

```
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().

```
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 grayscaled copy of an image.

```
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.

```
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.

```
len = myArray.length;
redValue = pixel.r;
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.

```
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 time to the array, meaning their size is also dynamic.

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

3.11.3 Element Retrieval

Array values are assigned and retrieved like so.

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

3.11.4 Length

Array size are retrieved using the length keyword.

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

3.12 Functions

Functions are declared using the following syntax.

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

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

```
func int add(int val1, int val2) {
    return val1 + val2;
}

#somewhere in main...
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.

```
    if(conditional statement is true){
        #do this
    }

    #using blocks
    if(x>0) {
        prints("x is positive");
    }
    elif(x<0) {
        prints("x is negative");
    }
    else { prints("x equals 0"); }
```

Loops are created as shown below.

```
    for( optional value initialization; conditional
    statement; optional increment) {
        #do stuff
    }

    while(conditional statement is true) {
        #do stuff
    }
```

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.

```
    int x = min(int_one, int_two);
    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.

```
    print(int);
    print(pint);
    prints(string);
    printf(float);
    printb(bool);
    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.

```
    Image img;
    img = load("edwards.png");
    #do some stuff to edwards
    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.

```
    Image load(string filename);
    void save(Image img, string filename);
    Image create( Int width, Int height, Pixel p);

    int width(Image img);
    int height(Image img);

    Pixel get_pixel(Image img, int x, int y);
    int set_pixel(Image img, int x, int y, Pixel p);

```

```
    Image image_add(Image img1, Image img2);
    Image image_subtract(Image img1, Image img2);

    Image image_invert(Image orig);
    Image image_paste(Image target, Image source, x, y);
    Image to_gray(Image orig);
    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.

```
func int maxElement(int[] inArray) {
    int max;
    max = 0;
    int i;
    for (i = 0; i < inArray.length; i = i + 1) {
        if (inArray[i] > max) {
            max = inArray[i];
        }
    }
    return max;
}
```

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.

```
func string flipAndGreyImage(string filePath) {
    Image testImage;
    testImage = load(filePath);
    testImage = flip(testImage);
    testImage = to_gray(testImage);
    string retPath;
    retPath = "newImage.png";
    Image newImage;
    save(testImage, retPath);
    return retPath;
}
```

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.

```
func Image halfHalf(Image image1, Image image2) { # alphas
    = 255 by default
    int i;
    int j;
    Pixel blankp;
    blankp = pixel(0, 0, 0, 0);
    for (i = 0; i < width(image1)/2; i = i + 1) {
        for (j = 0; j < width(image2); j = j + 1) {
            set_pixel(image1, i, j, blankp);
            set_pixel(image2, i+image1.width/2, j, blankp);
        }
    }
    Image image3;
    image3 = image1 + image2;
    return image3; #left side of image1, right side of
    image 2
}
```

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
```

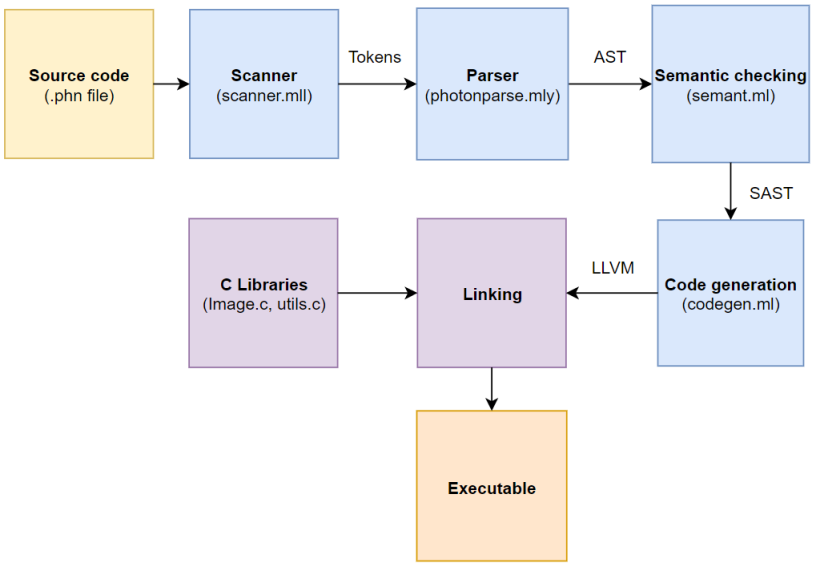
```
Sat Apr 24 15:44:55 2021 -0500 - Franky : Testing get_pixel.
Sat Apr 24 16:00:50 2021 -0400 - CalumMcCartan : add pixel alias
Sat Apr 24 15:39:01 2021 -0400 - CalumMcCartan : Colour aliases
Sat Apr 24 15:04:56 2021 -0400 - CalumMcCartan : add _red colour alias
Sat Apr 24 13:23:15 2021 -0400 - CalumMcCartan : 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 decleration 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 exhuastive 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 consistent 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. int + float = 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 familar.

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:

```
func int main()
{
    int[] a;
    a = [0,1,9,3,5];

    print(a[0]);
    print(a[1]);
    print(a[2]);

    print(a.length);

  return 0;
}
```

Generated LLVM:

```
; ModuleID = 'Photon'
source_filename = "Photon"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
define i32 @main() {
entry:
  %a = alloca { i32*, i32* }
  %array_size_ptr = getelementptr inbounds { i32*, i32* },
    { i32*, i32* }* %a, i32 0, i32 0
  %array_size = alloca i32
  store i32 0, i32* %array_size
  store i32* %array_size, i32** %array_size_ptr
  %list.arry = getelementptr inbounds { i32*, i32* }, {
    i32*, i32* }* %a, i32 0, i32 1
  %p = alloca i32, i32 1028
  store i32* %p, i32** %list.arry
  %new_array_ptr = alloca { i32*, i32* }
  %array_size_ptr1 = getelementptr inbounds { i32*, i32* },
    { i32*, i32* }* %new_array_ptr, i32 0, i32 0
  %array_size2 = alloca i32
  store i32 0, i32* %array_size2
  store i32* %array_size2, i32** %array_size_ptr1
  %list.arry3 = getelementptr inbounds { i32*, i32* }, {
    i32*, i32* }* %new_array_ptr, i32 0, i32 1
```

```
  %p4 = alloca i32, i32 1028
  store i32* %p4, i32** %list.arry3
  call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 0)
  call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 1)
  call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 9)
  call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 3)
  call void @array_addint({ i32*, i32* }* %new_array_ptr,
    i32 5)
  %new_array = load { i32*, i32* }, { i32*, i32* }*
    %new_array_ptr
  store { i32*, i32* } %new_array, { i32*, i32* }* %a
  %array_get = call i32 @array_getint({ i32*, i32* }* %a,
    i32 0)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get)
  %array_get5 = call i32 @array_getint({ i32*, i32* }* %a,
    i32 1)
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get5)
  %array_get7 = call i32 @array_getint({ i32*, i32* }* %a,
    i32 2)
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_get7)
  %array_size9 = call i32 @array_sizeint({ i32*, i32* }* %a)
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
    %array_size9)
  ret i32 0
}

declare i32 @printf(i8*, ...)
```

Load Images

Photon's source code:

```
func int main()
{
  Image img;
  int wid;
  int ht;

  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]
        c"Shapes.png\00"
10  @str.3 = private unnamed_addr constant [16 x i8]
        c"ShapesSaved.png\00"
11
12  define i32 @main() {
13  entry:
14    %img = alloca %PImage*
15    %wid = alloca i32
16    %ht = alloca i32
17    %load = call %PImage* @Image_load(i8* getelementptr
        inbounds ([11 x i8], [11 x i8]* @str, i32 0, i32 0))
18    store %PImage* %load, %PImage** %img
19    %img1 = load %PImage*, %PImage** %img
20    %width = call i32 @Image_width(%PImage* %img1)
21    store i32 %width, i32* %wid
22    %img2 = load %PImage*, %PImage** %img
23    %height = call i32 @Image_height(%PImage* %img2)
24    store i32 %height, i32* %ht
25    %wid3 = load i32, i32* %wid
26    %printf = call i32 (i8*, ...) @printf(i8* getelementptr
        inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
        %wid3)
27    %ht4 = load i32, i32* %ht
28    %printf5 = call i32 (i8*, ...) @printf(i8* getelementptr
        inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
        %ht4)
29    %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 test 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 directoy, 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

```
(* This file was copied from MicroC *)

(* Top-level of the Photon compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./photon.native [-a|-s|-l|-c] [file.phn]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Photonparse.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
```

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

11.1.2 scanner.mll

```
(* Ocamllex scanner for Photon *)

{ open Photonparse }

let digit = ['0' - '9']
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "#"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRACK }
| ']'      { RBRACK }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| '.'      { PERIOD }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "func"   { FUNC }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "pint"   { PINT }
| "bool"   { BOOL }
```

```
| "float"  { FLOAT }
| "void"   { VOID }
| "string" { STRING }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| "arr"     { ARRAY }
| "array_add" { ARRAY_ADD }
| "length"  { LENGTH }
| "Pixel"  { PIXEL }
| "Image"  { IMAGE }
| ('_' ['a'-'z']*) as str { ALIAS(str) }
| digits as lxm { LITERAL(int_of_string lxm) }
| '"' ([^ '"']* as str) '"'          { STRLIT(str) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as
    lxm { FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as
    lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^
    Char.escaped char)) }

and comment = parse
  "\n" { token lexbuf }
| _    { comment lexbuf }
```

11.1.3 ast.ml

```
(*
  Abstract Syntax Tree and functions for printing it
  Based on MicroC

  Authors:
  Calum McCartan (cm4114)
  Franky Campuzano (fc2608)
*)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq
    | Greater | Geq |
          And | Or

type uop = Neg | Not

type typ = Int | Pint | Bool | Float | Void | String |
    Array of typ | Image | Pixel

type bind = typ * string

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 ^ " " ^
      string_of_expr e2
88    | Unop(o, e) -> string_of_uop o ^ string_of_expr e
89    | Assign(v, e) -> v ^ " = " ^ string_of_expr e
90    | Call(f, el) ->
91        f ^ "(" ^ String.concat ", " (List.map string_of_expr
      el) ^ ")"
92    | Noexpr -> ""
93    | ArrayGet(id, e) -> "array_get " ^ id ^ ", " ^
      (string_of_expr e)
94    | ArraySize(id) -> "array_size " ^ id
95    | ArrayLiteral(_) -> "array_literal"
96    | Attr(i, a) -> i ^ "." ^ a
97
98  let rec string_of_stmt = function
99      Block(stmts) ->
100       "{\n" ^ String.concat "" (List.map string_of_stmt
      stmts) ^ "}\n"
101   | Expr(expr) -> string_of_expr expr ^ ";\n";
102   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
103   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^
      ")\n" ^ string_of_stmt s
104   | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
105       string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
106   | For(e1, e2, e3, s) ->
107       "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr
      e2 ^ " ; " ^
108       string_of_expr e3  ^ ") " ^ string_of_stmt s
109   | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
      string_of_stmt s
110   | ArraySet(id, e1, e2) -> "array_set " ^ id ^ ", " ^
      (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   /* Ocamlyacc 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
```

```
%token <string> ID FLIT STRLIT ALIAS
%token ARRAY
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { ([], [])               }
 | decls vdecl { (($2 :: fst $1), snd $1) }
 | decls fdecl { (fst $1, ($2 :: snd $1)) }

fdecl:
  FUNC typ ID LPAREN formals_opt RPAREN LBRACE fbody RBRACE
     { { typ = $2;
   fname = $3;
   formals = List.rev $5;
   locals = List.rev (fst $8);
   body = List.rev (snd $8) } }

fbody:
   /* nothing */ { ([], [])               }
 | fbody vdecl { (($2 :: fst $1), snd $1) }
 | fbody stmt { (fst $1, ($2 :: snd $1)) }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
    typ ID                   { [($1,$2)]     }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

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    | RETURN expr_opt SEMI                       { Return $2
              }
84    | LBRACE stmt_list RBRACE                    {
       Block(List.rev $2)    }
85    | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5,
       Block([])) }
86    | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5,
       $7)        }
87    | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
88                                               { For($3, $5,
       $7, $9)   }
89    | WHILE LPAREN expr RPAREN stmt           { While($3, $5)
              }
90    | ID LBRACK expr RBRACK ASSIGN expr SEMI        {
       ArraySet($1, $3, $6) }
91    | ARRAY_ADD LPAREN ID COMMA expr RPAREN SEMI    {
       ArrayAdd($3, $5) }
92
93
94  expr_opt:
95      /* nothing */ { Noexpr }
96    | expr          { $1 }
97
98  expr:
99      LITERAL          { Literal($1)            }
100   | FLIT             { Fliteral($1)           }
101   | STRLIT           { StrLiteral($1)}
102   | ALIAS            { Alias($1) }
103   | BLIT             { BoolLit($1)            }
104   | 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
       for printing it
3    Based on MicroC
4
5    Authors:
6    Akira Higaki (abh2171)
7    Calum McCartan (cm4114)
8    Franky Campuzano (fc2608)
9  *)
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
        successful,
18     throws an exception if something is wrong.
19
20     Check each global variable, then check each function *)
21
22  let check (globals, functions) =
23
24    (* Verify a list of bindings has no void types or
        duplicate names *)
25    let check_binds (kind : string) (binds : bind list) =
        List.iter (function
26       | (Void, b) -> raise (Failure ("illegal void " ^ kind ^
        " " ^ b))
27       | _ -> ()
28       ) binds;
29       let rec dups = function
30         | [] -> ()
31         | ((_,n1) :: (_,n2) :: _) when n1 = n2 -> raise
       (Failure ("duplicate " ^ kind ^ " " ^ n1))
32         | _ :: t -> dups t
33       in dups (List.sort (fun (_,a) (_,b) -> compare a b)
        binds)
34    in
35
36    (**** Check global variables ****)
37
```

```
38    check_binds "global" globals;
39
40    (**** Check functions ****)
41
42
43    (* Collect function declarations for built-in functions:
       no bodies *)
44    let built_in_decls =
45      let add_bind map (name, formals', rtype) =
       StringMap.add name {
46          (* object between brackets is func_decl object? *)
47          typ = rtype;
48          fname = name;
49          formals = formals';
50          locals = [];
51          body = []; (* empty list *)
52      } map
53      in List.fold_left add_bind StringMap.empty [
54        ("print", [(Int, "x")], Void);
55        ("printb", [(Bool, "x")], Void);
56        ("printf", [(Float, "x")], Void);
57        ("prints", [(String, "x")], Void);
58        ("min", [(Int, "x");(Int, "y")], Int);
59        ("max", [(Int, "x");(Int, "y")], Int);
60        ("sqrt", [(Float, "x")], Float);
61        ("load", [(String, "x")], Image);
62        ("save", [(Image, "img"); (String, "fname")], Void);
63        ("create", [(Int, "w"); (Int, "h"); (Pixel, "col")],
       Image);
64        ("destroy", [(Image, "img")], Void);
65        ("flip", [(Image, "img")], Image);
66        ("to_gray", [(Image, "img")], Image);
67        ("image_paste", [(Image, "target"); (Image, "orig");
       (Int, "x"); (Int, "y") ], Image);
68        ("image_invert", [(Image, "orig") ], Image);
69        ("image_add", [(Image, "img1"); (Image, "img2") ],
       Image);
70        ("image_subtract", [(Image, "img1"); (Image, "img2")
       ], Image);
71        ("get_pixel", [(Image, "img"); (Int, "x"); (Int,
       "y")], Pixel);
72        ("set_pixel", [(Image, "img"); (Int, "x"); (Int,
       "y"); (Pixel, "p");], Int);
73        ("width", [(Image, "img")], Int);
74        ("height", [(Image, "img")], Int);
75        ("pixel", [(Pint, "r"); (Pint, "g"); (Pint, "b");
       (Pint, "a")], Pixel);
76        ("pixel_attr", [(Pixel, "p"); (Int, "attr")], Pint)
77        ]
78    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
      exist"))
166         ) in expr (Call ("pixel", [PLiteral(r);
      PLiteral(g); PLiteral(b); PLiteral(a)]))
167       | BoolLit l  -> (Bool, SBoolLit l)
168       | Noexpr     -> (Void, SNoexpr)
169       | Id s       -> (type_of_identifier s, SId s)
170       | Attr(var, attr) ->
171           let lt = type_of_identifier var in
172           let err = var ^ " has no attribute " ^ attr in
173           (match lt with
174             | Pixel ->
175               (match attr with
176                 | "r" -> expr (Call("pixel_attr", [Id(var);
      Literal(0)]))
177                 | "g" -> expr (Call("pixel_attr", [Id(var);
      Literal(1)]))
178                 | "b" -> expr (Call("pixel_attr", [Id(var);
      Literal(2)]))
179                 | "a" -> expr (Call("pixel_attr", [Id(var);
      Literal(3)]))
180                 | _ -> raise (Failure (err)))
181             | Image ->
182                 (match attr with
183                 | "width" -> expr (Call("width", [Id(var)]))
184                 | "height" -> expr (Call("height",
      [Id(var)]))
185                 | _ -> raise (Failure (err)))
186             | _ -> raise (Failure (err)))
187       | Assign(var, e) as ex ->
188           let lt = type_of_identifier var
189           and (rt, e') = expr e in
190           let err = "illegal assignment " ^ string_of_typ
      lt ^ " = " ^
191             string_of_typ rt ^ " in " ^ string_of_expr ex
192           in (check_assign lt rt err, SAssign(var, (rt,
      e')))
193       | Unop(op, e) as ex ->
194           let (t, e') = expr e in
195           let ty = match op with
196             Neg when t = Int || t = Float -> t
197           | Not when t = Bool -> Bool
198           | _ -> raise (Failure ("illegal unary operator " ^
199                                  string_of_uop op ^
      string_of_typ t ^
200                                  " in " ^ string_of_expr
      ex))
201           in (ty, SUnop(op, (t, e')))
202       | Binop(e1, op, e2) as e ->
203           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
      param_length ^
246                             " arguments in " ^
      string_of_expr call))
247           else let check_call (ft, _) e =
248             let (et, e') = expr e in
249             let err = "illegal argument found " ^
      string_of_typ et ^
250               " expected " ^ string_of_typ ft ^ " in " ^
      string_of_expr e
251             in (check_assign ft et err, (et, e'))
252           in
253           let args' = List.map2 check_call fd.formals args
254           in (fd.typ, SCall(fname, args'))
255     in
256
257     let check_bool_expr e =
258       let (t', e') = expr e
259       and err = "expected Boolean expression in " ^
      string_of_expr e
260       in if t' != Bool then raise (Failure err) else (t',
      e')
261     in
262     let check_int_expr e =
263       let (t', e') = expr e
264       and err = "expected Integer expression in " ^
      string_of_expr e
265       in if t' != Int then raise (Failure err) else (t', e')
266     in
267
268     let check_match_array_type_expr l e =
269       let (t', e') as e'' = expr e in
270       let err = "array type and expression type do not
      match " ^ (string_of_typ t') ^ ", " ^ (string_of_sexpr
      e'') in
271       if t' != (check_array_type l) then raise (Failure
      err) else (t', e')
272     in
273
274     (* Return a semantically-checked statement i.e.
      containing sexprs *)
275     let rec check_stmt = function
276         Expr e -> SExpr (expr e)
277         | ArraySet (var, e1, e2) ->
278           SArraySet(check_array_type var, var,
      check_int_expr e1, check_match_array_type_expr var e2)
```

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

11.1.7 codegen.ml

```
(*
  Code generation: translate takes a semantically checked
    AST and produces LLVM IR
  Based on MicroC

  Authors:
  Akira Higaki (abh2171)
  Calum McCartan (cm4114)
  Franky Campuzano (fc2608)
  Phu D Pham (pdp2121)
*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Photon" in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and string_t   = L.pointer_type (L.i8_type context)
  and void_t     = L.void_type   context
  and image_t   = L.pointer_type (L.named_struct_type
    context "PImage")
  and pixel_t    = L.i32_type    context
  and array_t t   = L.struct_type context [| L.pointer_type
    (L.i32_type context); (L.pointer_type t) |]
  in

  (* Return the LLVM type for a Photon type *)
  let rec ltype_of_typ = function
      A.Int   -> i32_t
    | A.Pint  -> i8_t
    | A.Bool  -> i1_t
    | A.Float -> float_t
    | A.Void  -> void_t
    | A.String -> string_t
    | 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
      the_module) m
66      in List.fold_left global_var StringMap.empty globals
67    in
68
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
      i8_t |], "printf"
75        | "min" ->
76          L.function_type i32_t [| i32_t; i32_t |], "get_min"
77        | "max" ->
78          L.function_type i32_t [| i32_t; i32_t |], "get_max"
79        | "sqrt" ->
80          L.function_type float_t [| float_t |], "get_sqrt"
81        | "load" ->
82          L.function_type image_t [| string_t |], "Image_load"
83        | "save" ->
84          L.function_type i32_t [| image_t; string_t |],
      "Image_save"
85        | "create" ->
86          L.function_type image_t [| i32_t; i32_t; pixel_t
      |], "Image_create"
87        | "width" ->
88          L.function_type i32_t [| image_t |], "Image_width"
89        | "height" ->
90          L.function_type i32_t [| image_t |], "Image_height"
91        | "destroy" ->
92          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|],
      "Image_to_gray"
97        | "image_paste" ->
98          L.function_type image_t [| image_t; image_t; i32_t;
      i32_t|], "Image_paste"
99        | "image_invert" ->
100         L.function_type image_t [| image_t|], "Image_invert"
101       | "image_add" ->
102           L.function_type image_t [| image_t; image_t|],
      "Image_add"
103       | "image_subtract" ->
104           L.function_type image_t [| image_t; image_t|],
      "Image_subtract"
105       | "get_pixel" ->
106         L.function_type pixel_t [| image_t; i32_t; i32_t
      |], "get_pixel"
107       | "set_pixel" ->
108         L.function_type i32_t [| image_t; i32_t; i32_t;
      pixel_t |], "set_pixel"
109       | "pixel" ->
110         L.function_type pixel_t [| i8_t; i8_t; i8_t; i8_t;
      |], "pixel"
111       | "pixel_attr" ->
112         L.function_type i8_t [| pixel_t; i32_t |],
      "pixel_attr"
113       | _ ->
114         raise (Failure "internal error: built-in func does
      not exist ")
115     in
116     L.declare_function c_name func_t the_module
117   in
118
119   (* LLVM insists each basic block end with exactly one
      "terminator"
120        instruction that transfers control.  This function
      runs "instr builder"
121        if the current block does not already have a
      terminator.  Used,
122        e.g., to handle the "fall off the end of the
      function" case. *)
123
124   let add_terminal builder instr =
125     match L.block_terminator (L.insertion_block builder)
      with
126       | Some _ -> ()
127       | None -> ignore (instr builder)
128   in
```

```
129
130   (* array functions *)
131
132   (*
133   the following array code partially referenced from AP++
        2018 project:
134
        http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/reports/AP++.tar.gz
135   *)
136
137   (* ltype array_get(array a, i32_t index) *)
138   let array_get : L.llvalue StringMap.t =
139     let array_get_ty m typ =
140       let ltype = (ltype_of_typ typ) in
141
142       (* define the function type *)
143       let def_name = (type_str typ) in
144       let def = L.define_function ("array_get" ^ def_name)
        (L.function_type ltype [| L.pointer_type (array_t
        ltype); i32_t |]) the_module in
145
146       (* create array pointer *)
147       let build = L.builder_at_end context (L.entry_block
        def) in
148       let array_ptr = L.build_alloca (L.pointer_type
        (array_t ltype)) "array_ptr_alloc" build in
149       let _ = L.build_store (L.param def 0) array_ptr build
        in
150
151       (* create index pointer *)
152       let index_ptr = L.build_alloca i32_t "index_alloc"
        build in
153       let _ = L.build_store (L.param def 1) index_ptr build
        in
154
155       (* more building and allocating *)
156       let array_load = L.build_load array_ptr "array_load"
        build in
157       let array_ar_ptr = L.build_struct_gep array_load 1
        "array_ar_ptr" build in
158       let array_ar_load = L.build_load array_ar_ptr
        "array_load" build in
159
160       (* get return value *)
161       let index = L.build_load index_ptr "index_load" build
        in
162       let array_ar_elem_ptr = L.build_gep array_ar_load [|
        index |] "list_arry_element_ptr" build in
163       let ele_val = L.build_load array_ar_elem_ptr
        "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;
      A.Bool; A.Int; A.Float; A.String ] in
169
170     (* void array_set(array a, i32_t index, ltype value) *)
171     let array_set : L.llvalue StringMap.t =
172       let array_set_ty m typ =
173
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)
      (L.function_type void_t [| L.pointer_type (array_t
      ltype); i32_t; ltype |]) the_module in
177         let build = L.builder_at_end context (L.entry_block
      def) in
178
179         let array_ptr = L.build_alloca (L.pointer_type
      (array_t ltype)) "array_ptr_alloc" build in
180         ignore(L.build_store (L.param def 0) array_ptr build);
181
182         let array_load = L.build_load array_ptr "array_load"
      build in
183         let array_ar_ptr = L.build_struct_gep array_load 1
      "array_ar_ptr" build in
184         let array_ar_load = L.build_load array_ar_ptr
      "array_ar_load" build in
185
186         let index_element_ptr = L.build_gep array_ar_load [|
      L.param def 1 |] "array_ar_next_ele_ptr" build in
187         let _ = L.build_store (L.param def 2)
      index_element_ptr build in
188         let _ = L.build_ret_void build in
189         StringMap.add def_name def m in
190     List.fold_left array_set_ty StringMap.empty [ A.Pint;
      A.Bool; A.Int; A.Float; A.String ] in
191
192     (* void array_add(array, ltype value) *)
193     let array_add_ty m typ =
194       let ltype = (ltype_of_typ typ) in
195         let def_name = (type_str typ) in
196         let def = L.define_function ("array_add" ^ def_name)
      (L.function_type void_t [| L.pointer_type (array_t
      ltype); ltype |]) the_module in
197         let build = L.builder_at_end context (L.entry_block
      def) in
198         let array_ptr = L.build_alloca (L.pointer_type
      (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.arry" 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
      function_decls in
274     let builder = L.builder_at_end context (L.entry_block
      the_function) in
275
276     let int_format_str = L.build_global_stringptr "%d\n"
      "fmt" builder
277     and str_format_str = L.build_global_stringptr "%s\n"
      "fmt" builder
278     and float_format_str = L.build_global_stringptr "%g\n"
      "fmt" builder in
279
280     (* Construct the function's "locals": formal arguments
      and locally
281        declared variables.  Allocate each on the stack,
      initialize their
282        value, if appropriate, and remember their values in
      the "locals" map *)
283     let local_vars =
284       let add_formal m (t, n) p =
285         L.set_value_name n p;
286         let local = L.build_alloca (ltype_of_typ t) n
      builder in
287         ignore(
288           match t with
289             A.Array array_type -> init_array builder local
      array_type
290           | _ -> ()
291         );
292         ignore (L.build_store p local builder);
293         StringMap.add n local m
294         (* Allocate space for any locally declared
      variables and add the
295         * resulting registers to our map *)
296         and add_local m (t, n) =
297         let local_var = L.build_alloca (ltype_of_typ t) n
      builder in
298         ignore(
299           match t with
300             A.Array array_type -> init_array builder
      local_var array_type
301           | _ -> ()
302         );
303         StringMap.add n local_var m in
304         let formals = List.fold_left2 add_formal
      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.
          Check local names first, then global names *)
315     let lookup n =
316       try StringMap.find n local_vars with Not_found ->
          StringMap.find n global_vars
317     in
318
319     (* Clamp given integer between 0-255 so its ready to be
          cast to a pint *)
320     let build_pint_clamp e builder =
321       let max = (L.const_int i32_t 255) in
322       let max_cond = L.build_icmp L.Icmp.Sgt e max
          "compareMax" builder in
323       let e = L.build_select max_cond max e "selectMax"
          builder in
324       let min = (L.const_int i32_t 0) in
325       let min_cond = L.build_icmp L.Icmp.Slt e min
          "compareMin" builder in
326       L.build_select min_cond min e "selectMin" builder
327     in
328
329     (* Cast an evaluated expression 'e' from type 'rt' to
          type 'lt' *)
330     let cast_expr e lt rt builder =
331       if lt = rt then e else
332       let llt = ltype_of_typ lt in
333       match lt, rt with
334         | A.Pint, A.Int   -> let e' = build_pint_clamp e
          builder in L.build_trunc e' llt "pintCast" builder
335         | A.Int, A.Pint   -> L.build_zext e llt "intCast"
          builder
336         | A.Float, A.Pint -> L.build_uitofp e llt "floatCast"
          builder
337         | A.Float, A.Int  -> L.build_sitofp e llt "floatCast"
          builder
338         | _ -> raise (Failure "internal error: semant should
          have rejected an unsupported type conversion")
339     in
340
341     (* Construct code for an expression; return its value *)
342     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_fmul
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       in
399       (* If is pint op, then cast back from int to pint *)
400       if is_pint_op then cast_expr result A.Pint A.Int
      builder
401       else result
402     | SUnop(op, ((t, _) as e)) ->
403       (* Unop *)
404       let e' = expr builder e in (match op with
405           A.Neg when t = A.Float -> L.build_fneg
406         | A.Neg                 -> L.build_neg
407         | A.Not                 -> L.build_not) e' "tmp"
      builder
408
409     (* array functions *)
410     | SArrayGet (array_type, id, e) ->
411       L.build_call (StringMap.find (type_str array_type)
      array_get) [| (lookup id); (expr builder e) |]
      "array_get" builder
412     | SArraySize (array_type, id) ->
413       L.build_call ((StringMap.find (type_str array_type))
      array_size) [| (lookup id) |] "array_size" builder
414     | SArrayLiteral (array_type, literals) ->
415       let ltype = (ltype_of_typ array_type) in
416       let new_array_ptr = L.build_alloca (array_t ltype)
      "new_array_ptr" builder in
417       let _ = init_array builder new_array_ptr array_type in
418       let map_func literal =
419           ignore(L.build_call (StringMap.find (type_str
      array_type) array_add) [| new_array_ptr; (expr builder
      literal) |] "" builder);
420       in
421       let _ = List.rev (List.map map_func literals) in
422       L.build_load new_array_ptr "new_array" builder
423
424     | SCall (fname, f_args) ->
425       let cast_arg (lt, (rt, e)) =
426         let e' = expr builder (rt, e) in
427         cast_expr e' lt rt builder
428       in
```

```
429         let args = Array.of_list (List.rev (List.map
    (cast_arg) (List.rev f_args))) in
430         let (fdef, args', result) = match fname with
431           (* Built in functions with modifed arguments *)
432           | "printb"
433           | "print"    -> (func_decl "printf"), [|
    int_format_str ; args.(0) |],   "printf"
434           | "printf"   -> (func_decl "printf"), [|
    float_format_str ; args.(0) |], "printf"
435           | "prints"   -> (func_decl "printf"), [|
    str_format_str ; args.(0) |],   "printf"
436           (* Built in functions with unmodifed arguments *)
437           | "min" | "max" | "sqrt" | "load" | "save" |
    "create" | "width"
438           | "height" | "destroy" | "flip" | "to_gray" |
    "image_paste" | "image_invert"| "image_add"|
    "image_subtract"
439           | "get_pixel" | "set_pixel" | "pixel" | "pixel_attr"
440             -> (func_decl fname), args, fname
441           (* User defined function *)
442           | _ ->
443               let (fdef, fdecl) = StringMap.find fname
    function_decls in
444               let result = (match fdecl.styp with
445                 | A.Void -> ""
446                 | _ -> fname ^ "_result")
447               in fdef, args, result
448         in
449         L.build_call fdef args' result builder
450     in
451
452
453     (* Build the code for the given statement; return the
      builder for
454         the statement's successor (i.e., the next instruction
      will be built
455         after the one generated by this call) *)
456
457     let rec stmt builder = function
458       | SBlock sl -> List.fold_left stmt builder sl
459       | SArrayAdd (id, e) ->
460           ignore(L.build_call (StringMap.find (type_str (fst
      e)) array_add) [| (lookup id); (expr builder e) |] ""
      builder); builder
461       | SArraySet (array_type, id, e1, e2) ->
462           ignore(L.build_call (StringMap.find (type_str
      array_type) array_set) [| (lookup id); (expr builder
      e1); (expr builder e2) |] "" builder); builder
463       | SExpr e -> ignore(expr builder e); builder
464       | 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"
    the_function in
476         let build_br_merge = L.build_br merge_bb in (*
    partial function *)
477
478         let then_bb = L.append_block context "then"
    the_function in
479         add_terminal (stmt (L.builder_at_end context
    then_bb) then_stmt)
480           build_br_merge;
481
482         let else_bb = L.append_block context "else"
    the_function in
483         add_terminal (stmt (L.builder_at_end context
    else_bb) else_stmt)
484           build_br_merge;
485
486         ignore(L.build_cond_br bool_val then_bb else_bb
    builder);
487         L.builder_at_end context merge_bb
488
489       | SWhile (predicate, body) ->
490         let pred_bb = L.append_block context "while"
    the_function in
491         ignore(L.build_br pred_bb builder);
492
493         let body_bb = L.append_block context "while_body"
    the_function in
494         add_terminal (stmt (L.builder_at_end context body_bb)
    body)
495           (L.build_br pred_bb);
496
497         let pred_builder = L.builder_at_end context pred_bb in
498         let bool_val = expr pred_builder predicate in
499
500         let merge_bb = L.append_block context "merge"
    the_function in
501         ignore(L.build_cond_br bool_val body_bb merge_bb
    pred_builder);
502         L.builder_at_end context merge_bb
```

```
503
504       (* Implement for loops as while loops *)
505       | SFor (e1, e2, e3, body) ->
506         stmt builder ( SBlock [SExpr e1 ; SWhile (e2, SBlock
      [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
       library.
3  Image_load, Image_free and Image_save are written by the
       stb_image
4  library team, the rest is written by the Photon team.
5
6  Authors:
7  Franky Campuzano (fc2608)
8  Akira Higaki (abh2171)
9  Calum McCartan (cm4114)
10 Phu D Pham (pdp2121)
11 */
12
13 #include "Image.h"
14 #include "utils.h"
15 #include <math.h>
16
17 #define STB_IMAGE_IMPLEMENTATION
18 #include "stb_image/stb_image.h"
19 #define STB_IMAGE_WRITE_IMPLEMENTATION
20 #include "stb_image/stb_image_write.h"
21
22 Image* Image_load(const char *fname) {
23     Image* img = malloc(sizeof(Image));
```

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

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

```
113     }
114     return img;
115 }
116 
117 void Image_free(Image *img) {
118     if(img->allocation_ != NO_ALLOCATION && img->data !=
        NULL) {
119         if(img->allocation_ == STB_ALLOCATED) {
120             stbi_image_free(img->data);
121         } else {
122             free(img->data);
123         }
124         img->data = NULL;
125         img->width = 0;
126         img->height = 0;
127         img->size = 0;
128         img->allocation_ = NO_ALLOCATION;
129     }
130 }
131 
132 Image* Image_paste( Image *gray, const Image *orig, int x,
        int y ) {
133     //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
        orig->channels >= 3), "The input image must have at
        least 3 channels.");
134 
135     int pos = get_position(gray->width, gray-> channels,
        x,y);
136 
137     int i = 0;
138     bool fixedpos = false;
139     for(unsigned char *p = orig->data, *pg = gray->data; p
        != orig->data + orig->size; p += orig->channels, pg +=
        gray->channels) {
140 
141         if(!fixedpos) {
142             pg+=pos;
143             fixedpos = true;
144         }
145 
146         if(i % ((orig-> width)*4) == 0 && i!= 0) {
147             pg+= (((gray-> width)*4) - (orig->width *4));
148 
149         }
150 
151         *pg = *p;
152         *(pg + 1) = *(p+1);
153         *(pg + 2) = *(p+2);
154 
155         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 &&
    orig->channels >= 3), "The input image must have at
    least 3 channels.");
166
167     uint8_t pix;
168     int8_t maxval = 255;
169     int i = 0;
170     for(unsigned char *p = orig->data; p != orig->data +
    orig->size; p += orig->channels) {
171         pix = *p;
172         *p = maxval - pix;
173
174         pix = *(p+1);
175         *(p + 1) = maxval - pix;
176
177         pix = *(p+2);
178         *(p + 2) = maxval - pix;
179     }
180     return orig;
181 }
182
183 Image* Image_add( Image *img1, Image *img2) {
184     int flag, w, h;
185     w = h = 1;
186     flag = img1->size >= img2->size ? 1 : 2;
187     int i = 0;
188     Image* gray;
189     if (flag == 1){
190         gray = Image_create(img1->width, img1->height,
    pixel(0, 0, 0, 255));
191         for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
    g_idx < gray->size; g_idx+= gray->channels, img1_idx +=
    img1->channels) {
192             for (int c = 0; c < img1->channels; c++) {
193                 gray->data[g_idx + c] = img1->data[img1_idx
    + c];
194             }
195             if (w <= img2->width && h <= img2->height) {
196                 for (int c = 0; c < img2->channels; c++) {
197                     gray->data[g_idx + c] +=
    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,
    pixel(0, 0, 0, 255));
210         for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
    g_idx < gray->size; g_idx+= gray->channels, img2_idx +=
    img2->channels) {
211             //printf("%d", g_idx);
212             for (int c = 0; c < img2->channels; c++) {
213                 gray->data[g_idx + c] = img2->data[img2_idx
    + c];
214             }
215             if (w <= img1->width && h <= img1->height) {
216                 for (int c = 0; c < img1->channels; c++) {
217                     gray->data[g_idx + c] +=
    img1->data[img1_idx + c];
218                 }
219                 img1_idx += img1->channels;
220             }
221             if (w == gray->width) {
222                 w = 1;
223                 h += 1;
224             } else {
225                 w += 1;
226             }
227         }
228     }
229     return gray;
230 }
231
232 Image* Image_subtract( Image *img1, Image *img2) {
233     int flag, w, h;
234     w = h = 1;
235     flag = img1->size >= img2->size ? 1 : 2;
236     int i = 0;
237     Image* gray;
238     if (flag == 1){
239         gray = Image_create(img1->width, img1->height,
    pixel(0, 0, 0, 255));
240         for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
    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

```
/*
Image headers for Photon that uses the stb_image library.
Defines the Image struct, its pointer PImage, the Pixel struct, and its
PPixel pointer.
struct Image, Image_load, Image_free and Image_save are written by the
stb_image library team. The rest is written by the Photon team.

Authors:
Akira Higaki (abh2171)
Calum McCartan (cm4114)
*/

#pragma once

#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <math.h>

enum allocation_type {
    NO_ALLOCATION, SELF_ALLOCATED, STB_ALLOCATED
};

typedef struct {
    int width;
    int height;
    int channels;
    size_t size;
    uint8_t *data;
    enum allocation_type allocation_;
} Image;

typedef struct {
    uint8_t r;
    uint8_t g;
    uint8_t b;
    uint8_t a;
} Pixel;

typedef struct {
    Image img;
} *PImage;

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()
```

```
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif
```

11.1.11 utils.h

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

// Error utility macro
#define ON_ERROR_EXIT(cond, message) \
do { \
    if((cond)) { \
        printf("Error in function: %s at line %d\n",
    __func__, __LINE__); \
        perror((message)); \
        exit(1); \
    } \
} while(0)

// Check if a string "str" ends with a substring "ends"
static inline bool str_ends_in(const char *str, const char
    *ends) {
    char *pos = strrchr(str, '.');
    return !strcmp(pos, ends);
}
```

11.1.12 Makefile

```
# "make test" Compiles everything and runs the regression
    tests

.PHONY : test
test : all testall.sh
  ./testall.sh

retest:
  dos2unix testall.sh
  make clean
  clear
```

```
  make test

# "make all" builds the executable as well as the built-in library designed
# to test linking external code

.PHONY : all
all : photon.native utils.o Image.o

# "make photon.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

photon.native :
  opam config exec -- \
  ocamlbuild -use-ocamlfind photon.native

# "make clean" removes all generated files

.PHONY : clean
clean :
  rm -rf testall.log ocamlllvm *.diff _build photon.native utils.o Image.o images-out
  ocamlbuild -clean

utils : utils.c
  cc -o utils -DBUILD_TEST utils.c

# Building the tarball

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags photon.ml photonparse.mly \
  README scanner.mll semant.ml testall.sh \
  utils.c Image.c utils.h Image.h arcade-font.pbm font2c \
  Dockerfile

photon.tar.gz : $(TARFILES)
  cd .. && tar czf photon/photon.tar.gz \
    $(TARFILES:%=photon/%)
```

11.1.13 testall.sh

```
#!/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
```

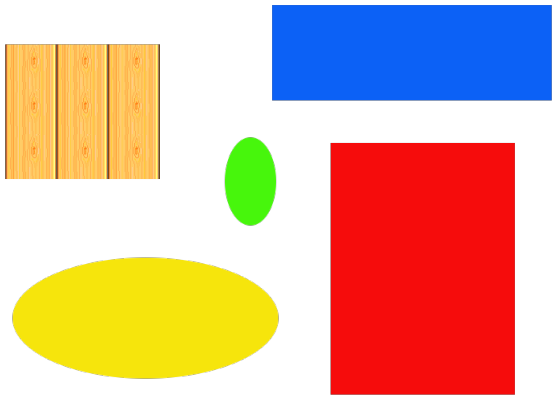
```
99      echo 1>&2
100     echo "###### Testing $basename" 1>&2
101
102     generatedfiles=""
103
104     generatedfiles="$generatedfiles ${basename}.ll
        ${basename}.s ${basename}.exe ${basename}.out" &&
105     Run "$Photon" "$1" ">" "${basename}.ll" &&
106     Run "$LLC" "-relocation-model=pic"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

```
144
145       if [ $error -eq 0 ] ; then
146     if [ $keep -eq 0 ] ; then
147         rm -f $generatedfiles
148     fi
149     echo "OK"
150     echo "###### SUCCESS" 1>&2
151       else
152     echo "###### FAILED" 1>&2
153     globalerror=$error
154       fi
155   }
156
157   while getopts kdpsh c; do
158       case $c in
159     k) # Keep intermediate files
160         keep=1
161         ;;
162     h) # Help
163         Usage
164         ;;
165       esac
166   done
167
168   shift `expr $OPTIND - 1`
169
170   LLIFail() {
171     echo "Could not find the LLVM interpreter \"$LLI\"."
172     echo "Check your LLVM installation and/or modify the LLI
        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 ]
```

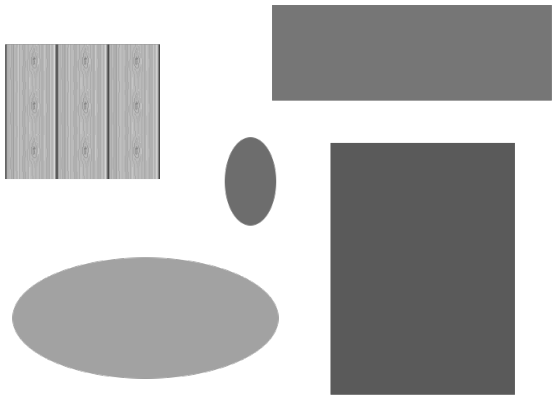
```
193 then
194     files=$@
195 else
196     files="tests/test-*.phn tests/fail-*.phn"
197 fi
198 
199 for file in $files
200 do
201     case $file in
202   *test-*)
203       Check $file 2>> $globallog
204       ;;
205   *fail-*)
206       CheckFail $file 2>> $globallog
207       ;;
208   *)
209       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
```

```
  return 0;
}
```

fail-array2.phn

```
func int main()
{
    string[] a;
    a = [0,1,9,3];
    print(a.length);

  return 0;
}
```

fail-assign1.phn

```
func int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; # Fail: assigning a bool to an integer
}
```

fail-assign2.phn

```
func int main()
{
  int i;
  bool b;

  b = 48; # Fail: assigning an integer to a bool
}
```

fail-assign3.phn

```
func void myvoid()
{
  return;
}
```

```
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

```
func int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; # Error: code after a return
}
```

fail-expr1.phn

```
int a;
bool b;

func void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; # Error: bool + int
}

func int main()
{
  return 0;
}
```

fail-expr2.phn

```
int a;
bool b;

func void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; # Error: bool + int
}

func int main()
{
```

```
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;
```

```
}
```

fail-for3.phn

```
func int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} # i is an integer, not
    Boolean

  return 0;
}
```

fail-for4.phn

```
func int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} # j undefined

  return 0;
}
```

fail-for5.phn

```
func int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); # Error: no function foo
  }

  return 0;
}
```

fail-func1.phn

```
func int foo() {}

func int bar() {}

```

```
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;
```

```
}
```

fail-func5.phn

```
func int foo() {}

func int bar() {
  int a;
  void b; # Error: illegal void local b
  bool c;

  return 0;
}

func int main()
{
  return 0;
}
```

fail-func6.phn

```
func void foo(int a, bool b)
{
}

func int main()
{
  foo(42, true);
  foo(42); # Wrong number of arguments
}
```

fail-func7.phn

```
func void foo(int a, bool b)
{
}

func int main()
{
  foo(42, true);
  foo(42, true, false); # Wrong number of arguments
}
```

fail-func8.phn

```
func void foo(int a, bool b)
{
}

func void bar()
{
}

func int main()
{
  foo(42, true);
  foo(42, bar()); # int and void, not int and bool
}
```

fail-func9.phn

```
func void foo(int a, bool b)
{
}

func int main()
{
  foo(42, true);
  foo(42, 42); # Fail: int, not bool
}
```

fail-getpixel1.phn

```
func int main()
{
  #whatever
  Image img;

  int index;
  Pixel p;
  float x;
  int y;
  int position;
  int channels;
  Pixel ps;

  img = load("edwards.png");

  #non integer position should return an error
  x = 1.5;
  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

```
func int main()
{
  if (true) {
    foo; # Error: undeclared variable
  }
}
```

fail-if3.phn

```
func int main()
{
  if (true) {
    42;
  } else {
    bar; # Error: undeclared variable
  }
}
```

fail-imageadd.phn

```
func int main() {

    Image img1;
    img1 = load("Shapes.png");

    img1 + 5;

    return 0;
}
```

fail-imagecreate1.phn

```
func int main()
{
  Image img;
  int wid;
  int ht;

  pint r;
  pint g;

  r = g = 0;
  #missing b and a layers
  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
```

```
  wid = width(img);
  ht = height(img);
  print(wid);
  print(ht);

  save(img, "ShapesSaved.png");

  return 0;
}
```

fail-imagepaste1.phn

```
func int main()
{
  Image imgshapes;
  Image imgedwards;
  Image newimg;
  Image testimg;
  int wid;
  int ht;

  imgshapes = load("Shapes.png");
  imgedwards = load("edwards.png");

  newimg = image_paste(imgshapes, ht,0,0);

  wid = width(newimg);
  ht = height(newimg);
  print(wid);
  print(ht);

  save(newimg, "ImgPasteTest.png");

  return 0;
}
```

fail-min-max1.phn

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

fail-missingattr.phn

```
func int main() {

    Image img1;
    img1 = load("Shapes.png");

    img1.cheese;

}
```

fail-nomain.phn

fail-pint1.phn

```
func int main()
{
    pint x;
    x = 20.25;

    print(x);

    return 0;
}
```

fail-print.phn

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

fail-printb.phn

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

fail-printhello1.phn

```
func int main()
{
  #whatever
  prints(hello world);
  return 0;
}
```

fail-return1.phn

```
func int main()
{
  return true; # Should return int
}
```

fail-return2.phn

```
func void foo()
{
  if (true) return 42; # Should return void
  else return;
}

func int main()
{
  return 42;
}
```

fail-setpixel1.phn

```
func int main()
{
  #testing set_pixel
  Image img;

  int index;
  int position;
  int channels;

  int p1;
  int x;
  int y;

  img = load("Shapes.png");

  #for gettign a specific pixel, input the width * height
    coordinate you want
  x = 0;
  y = 0;
  #set pixel to an int should return error
  p1 = 30;

  # Set pixel
  set_pixel(img, x, y, p1);
  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

```
func int add(int x, int y)
{
  return x + y;
}

func int main()
{
  print( add(17, 25) );
  return 0;
}
```

test-arith1.phn

```
func int main()
{
  print(39 + 3);
  return 0;
}
```

test-arith2.phn

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

test-arith3.phn

```
func int foo(int a)
{
  return a;
}

func int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}
```

test-array1.phn

```
func int main()
{
    int[] a;
    a = [0,1,2];

    print(a[0]);
    print(a[1]);
    print(a[2]);

  return 0;
}
```

test-array2.phn

```
func int main()
{
    string[] a;
    a = ["dog", "cat", "fish"];

    prints(a[0]);
    prints(a[1]);
    prints(a[2]);

  return 0;
}
```

test-array3.phn

```
func int main()
{
    float[] a;
    a = [1.1,2.2,3.3];

    printf(a[0]);
    printf(a[1]);
    printf(a[2]);

  return 0;
}
```

test-array4.phn

```
func int main()
{
```

```
    int[] a;
    a = [0,1,9,3,5];

    print(a[0]);
    print(a[1]);
    print(a[2]);

    print(a.length);

  return 0;
}
```

test-array5.phn

```
func int main()
{
    int[] a;
    int i;

    a = [0,1,9,3,6];

    print(a[0]);
    print(a[1]);
    print(a[2]);

    print(a.length);

    a[0] = 21;
    print(a[0]);


    for (i = 0 ; i < a.length; i = i + 1) {
    print(a[i]);
  }

    print(a.length);

  return 0;
}
```

test-colour-alias.phn

```
func int main()
{
    Pixel a;
    a = _red;

```

```
    print(a.r);
    print(a.g);
    print(a.b);
    print(a.a);

    return 0;
}
```

test-decl-order.phn

```
func int main()
{
    int x;
    x = 12;
    print(x);

    y = 5;
    int y; # vdecls are hoisted

    int z;
    z = x + y;
    print(z);

    return 0;
}
```

test-fib.phn

```
func int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

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

test-float1.phn

```
func int main()
{
  float a;
  a = 3.14159267;
  printf(a);
  return 0;
}
```

test-float2.phn

```
func int main()
{
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  printf(c);
  return 0;
}
```

test-float3.phn

```
func void testfloat(float a, float b)
{
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);
}

func int main()
{
  float c;
  float d;

```

```
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-func1.phn

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

```
  a = add(39, 3);
  print(a);
  return 0;
}
```

test-func2.phn

```
# Bug noticed by Pin-Chin Huang

func int fun(int x, int y)
{
  return 0;
}

func int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);
  return 0;
}
```

test-func3.phn

```
func void printem(int a, int b, int c, int d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

func int main()
{
  printem(42,17,192,8);
  return 0;
}
```

test-func4.phn

```
func int add(int a, int b)
{
  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()
```

```
{
  foo(73);
  print(a);
  return 0;
}
```

test-func8.phn

```
func void foo(int a)
{
  print(a + 3);
}

func int main()
{
  foo(40);
  return 0;
}
```

test-func9.phn

```
func void foo(int a)
{
  print(a + 3);
  return;
}

func int main()
{
  foo(40);
  return 0;
}
```

test-gcd.phn

```
func int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

func int main()
{
  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);
```

```
  print(p.b);
  print(p.a);

  return 0;
}
```

test-global1.phn

```
int a;
int b;

func void printa()
{
  print(a);
}

func void printbb()
{
  print(b);
}

func void incab()
{
  a = a + 1;
  b = b + 1;
}

func int main()
{
  a = 42;
  b = 21;
  printa();
  printbb();
  incab();
  printa();
  printbb();
  return 0;
}
```

test-global2.phn

```
bool i;

func int main()
{
  int i; # Should hide the global i

```

```
  i = 42;
  print(i + i);
  return 0;
}
```

test-global3.phn

```
int i;
bool b;
int j;

func int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
```

test-hello.phn

```
func int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}
```

test-if1.phn

```
func int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
```

test-if2.phn

```
func int main()
{
  if (true) print(42); else print(8);
  print(17);
```

```
  return 0;
}
```

test-if3.phn

```
func int main()
{
  if (false) print(42);
  print(17);
  return 0;
}
```

test-if4.phn

```
func int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}
```

test-if5.phn

```
func int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

func int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

test-if6.phn

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

```
  int x;
  x = 10;
  if (b)
    if (x == 10)
      x = 42;
  else
    x = 17;
  return x;
}

func int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

test-imageadd1.phn

```
func int main()
{
  Image imgshapes;
  Image imgedwards;
  Image newimg;
  int wid;
  int ht;

  imgshapes = load("Shapes.png");
  imgedwards = load("edwards.png");

  newimg = image_add(imgshapes, imgedwards);

  wid = width(newimg);
  ht = height(newimg);
  print(wid);
  print(ht);

  save(newimg, "ImgAddTest.png");


  # Using operators
  Image newimg2;
  newimg2 = imgshapes + imgedwards;
  save(newimg2, "ImgAddTest2.png");
  print(newimg2.width);
  print(newimg2.height);

  return 0;
```

```
}
```

test-imageattr.phn

```
func int main() {
    Image img;
    img = load("Shapes.png");

    print(img.width);
    print(img.height);

    return 0;
}
```

test-imagecreate.phn

```
func int main()
{
  Image img;
  int wid;
  int ht;

  img = create(800, 600, _blue);

  wid = width(img);
  ht = height(img);
  print(wid);
  print(ht);

  save(img, "blueImgTest.png");

  return 0;
}
```

test-imagedestroy.phn

```
func int main()
{
  Image img;
  int wid;
  int ht;

  img = load("Shapes.png");

  wid = width(img);
  ht = height(img);
```

```
  print(wid);
  print(ht);

  destroy(img);

  wid = width(img);
  ht = height(img);
  print(wid);
  print(ht);

  return 0;
}
```

test-imageflip.phn

```
func int main()
{
  Image img;
  Image flippedimg;
  int wid;
  int ht;

  img = load("Shapes.png");
  flippedimg = flip(img);

  wid = width(flippedimg);
  ht = height(flippedimg);
  print(wid);
  print(ht);

  save(flippedimg, "flipImgTest.png");

  return 0;
}
```

test-imagegray.phn

```
func int main()
{
  Image img;
  Image grayimg;
  int wid;
  int ht;

  img = load("Shapes.png");
  grayimg = to_gray(img);

```

```
  wid = width(grayimg);
  ht = height(grayimg);
  print(wid);
  print(ht);

  save(grayimg, "grayImgTest.png");

  return 0;
}
```

test-imageinvert.phn

```
func int main()
{
  Image imgedwards;
  Image newimg;
  int wid;
  int ht;

  imgedwards = load("edwards.png");

  newimg = image_invert(imgedwards);

  wid = width(newimg);
  ht = height(newimg);
  print(wid);
  print(ht);

  save(newimg, "ImgInvertTest.png");

  return 0;
}
```

test-imageload1.phn

```
func int main()
{
  Image img;
  int wid;
  int ht;

  img = load("Shapes.png");

  wid = width(img);
  ht = height(img);
  print(wid);
  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;
```

```
    int y;


    imgshapes = load("Shapes.png");
    imgedwards = load("edwards.png");


    x = 300;
    y = 300;
    newimg = image_paste(imgshapes, imgedwards, x,y);

    wid = width(newimg);
    ht = height(newimg);
    print(wid);
    print(ht);

    save(newimg, "ImgPasteTest2.png");

    return 0;
}
```

test-imagesubtract1.phn

```
func int main()
{
    Image imgshapes;
    Image imgedwards;
    Image newimg;
    int wid;
    int ht;

    imgshapes = load("Shapes.png");
    imgedwards = load("edwards.png");

    newimg = image_subtract(imgshapes, imgedwards);

    wid = width(newimg);
    ht = height(newimg);
    print(wid);
    print(ht);

    save(newimg, "ImgSubtractTest.png");


    # Using operators
    Image newimg2;
    newimg2 = imgshapes - imgedwards;
```

```
  save(newimg2, "ImgSubtractTest2.png");
  print(newimg2.width);
  print(newimg2.height);

  return 0;
}
```

test-int-to-float.phn

```

func int main()
{
    int i;
    float f;
    i = 5;
    f = 1.7;
    f = 1.2 + 5;
    printf(f);
    printf(6.4 / 2);
    printf(2 * 2.2);
    printf(2 + 2.2);
    printb(2 > 2.2);
    printb(3.14 != 3);
}
```

test-local1.phn

```
func void foo(bool i)
{
  int i; # Should hide the formal i

  i = 42;
  print(i + i);
}

func int main()
{
  foo(true);
  return 0;
}
```

test-local2.phn

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

```
  bool d;

  c = a;

  return c + 10;
}

func int main() {
 print(foo(37, false));
 return 0;
}
```

test-min-max1.phn

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

test-min-max2.phn

```
func int main()
{
    int a;
    int b;
    a = 1;
    b = 2;
    print (min(a,b));
    a = 3;
    print (max(a,b));
    return 0;
}
```

test-mixed-numeric-types.phn

```
func int main()
{

    int i;
    pint p;
    float f;

    p = 10;
    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) {
```

```
}
```

test-ops1.phn

```
func int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

test-ops2.phn

```
func int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
```

```
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}
```

test-pint-clamp.phn

```
func int main()
{
    pint x;
    int i;

    x = 42;
    print(x);
    x = 255;
    print(x);
    x = 256;
    print(x);
    x = 1000;
    print(x);
    x = 0;
    print(x);
    x = -1;
    print(x);

    i = 10;
    x = 1 + 1;
    print(x);

    pint x2;
    x = 200;
    x2 = 200;
    print(x + x2);

    x = getBigInt();
    print(x);

    x = x2;
    printb(x == x2);

    printPint(66);
    printPint(-12);

    return 0;
}

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

```
func int main()
{
  #whatever
  prints("hello world");
  return 0;
}
```

test-setpixel1.phn

```
func int main()
{
  #testing set_pixel
  Image img;

  int index;
  int position;
  int channels;

  Pixel p1;
  int x;
  int y;

  img = load("Shapes.png");

  #for gettign a specific pixel, input the width * height
    coordinate you want
  x = 0;
  y = 0;
  p1 = pixel(255, 0, 0, 255);

  # Set pixel
  set_pixel(img, x, y, p1);
  save(img, "ShapesSavedMod.png");

  # Check pixel was changed
  img = load("ShapesSavedMod.png");
  Pixel p2;
  p2 = get_pixel(img, x, y);

  print(p2.r);
  print(p2.g);
  print(p2.b);
  print(p2.a);

  return 0;
}
```

test-sqrt1.phn

```
func int main()
{
  printf(sqrt(4));
  printf(sqrt(2.25));
  printf(sqrt(-1));
  return 0;
}
```

test-sqrt2.phn

```
func int main()
{
  float a;
  a = 9.0;
  float b;
  b = 0.25;
  printf(sqrt(a));
  printf(sqrt(b));
}
```

test-var1.phn

```
func int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}
```

test-var2.phn

```
int a;

func void foo(int c)
{
  a = c + 42;
}

func int main()
{
  foo(73);
  print(a);
  return 0;
```

```
}
```

test-while1.phn

```
func int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

test-while2.phn

```
func int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

func int main()
{
  print(foo(7));
  return 0;
}
```