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

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

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.

```
make all
./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.

```c
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>An integer</td>
</tr>
<tr>
<td>float</td>
<td>A floating-point number</td>
</tr>
<tr>
<td>string</td>
<td>A sequence of characters</td>
</tr>
<tr>
<td>bool</td>
<td>True or False values</td>
</tr>
<tr>
<td>pint</td>
<td>A special type of Int whose value can only range from 0 to 255</td>
</tr>
</tbody>
</table>

3.1.2 Pint Type

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

```plaintext
pint p;
pint q;
p = 150;
q = 200;
print(p + q); # prints 255
print(p - q); # prints 0
```
3.1.3 Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>A single unit of multiple grouped values</td>
</tr>
<tr>
<td>Pixel</td>
<td>A group of four Pint values</td>
</tr>
<tr>
<td>Image</td>
<td>A one dimensional array of pixel values.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Required type</th>
<th>Accepted types</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>int, pint</td>
</tr>
<tr>
<td>pint</td>
<td>int, pint</td>
</tr>
<tr>
<td>float</td>
<td>int, pint, float</td>
</tr>
</tbody>
</table>

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.

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

Below are keywords reserved for data types and structures in the language.
Color aliases are also keywords reserved by the language - which can be referenced using a ‘\_', such as _black

<table>
<thead>
<tr>
<th>Alias</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_black</td>
<td>Pixel(0, 0, 0, 255)</td>
</tr>
<tr>
<td>_white</td>
<td>Pixel(255, 255, 255, 255)</td>
</tr>
<tr>
<td>_gray</td>
<td>Pixel(128, 128, 128, 255)</td>
</tr>
<tr>
<td>_red</td>
<td>Pixel(255, 0, 0, 255)</td>
</tr>
<tr>
<td>_green</td>
<td>Pixel(0, 255, 0, 255)</td>
</tr>
<tr>
<td>_blue</td>
<td>Pixel(0, 0, 255, 255)</td>
</tr>
<tr>
<td>_cyan</td>
<td>Pixel(0, 255, 255, 255)</td>
</tr>
<tr>
<td>_magenta</td>
<td>Pixel(255, 0, 255, 255)</td>
</tr>
<tr>
<td>_yellow</td>
<td>Pixel(255, 255, 0, 255)</td>
</tr>
</tbody>
</table>

### 3.5 Operators & Logical Expressions

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

### 3.6 Precedence
### 3.7 Syntax

#### 3.7.1 Variable Declaration & Assignment

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

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

For example...

```c
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.

```c
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.

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

### 3.7.4 String literals

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

```plaintext
1. string x;
2. x = "hello ´friend´";
3. printf("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.

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

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

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

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

```plaintext
1. x = myPixel.r;
2. y = myPixel.g;
```
Unlike the Image type, Pixels are passed by value rather than by reference and so do not need to be manually destroyed.

3.9 Image

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

3.9.1 Creating an image

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

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

3.9.2 Width and height

The width and height attribute is accessible.

```c
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.

```c
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:

```c
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:

```c
newImg = img1 + img2;
```
Images are added with the following equation:

\[
\text{newImg.pixels}[x][y].red = \text{img1.pixels}[x][y].red \times \text{img1.pixel}[x][y].alpha / 255 + \text{img2.pixels}[x][y].red \times \text{img2.pixel}[x][y].alpha / 255
\]

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:

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

Images are subtracted with the following equation:

\[
\text{newImg.pixel}[x][y].red = \text{img1.pixel}[x][y].red \times \text{img1.pixel}[x][y].alpha / 255 - \text{img2.pixel}[x][y].red \times \text{img2.pixel}[x][y].alpha / 255
\]

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:

```c
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:

```c
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.

```plaintext
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().

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
string[] greetings;
```

3.11.2 Size

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

```plaintext
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.

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

3.11.4 Length

Array size are retrieved using the length keyword.

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

3.12 Functions

Functions are declared using the following syntax.

```plaintext
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.

```bash
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.

```bash
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.

```bash
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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);
```
3.16 Example Code

3.16.1 Maximum of an array

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

```c
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.

```c
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 image2
}
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

<table>
<thead>
<tr>
<th>Date</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 3</td>
<td>Language proposal submitted</td>
</tr>
<tr>
<td>February 16</td>
<td>Git repo created, first commit</td>
</tr>
<tr>
<td>February 24</td>
<td>LRM and Parser finished</td>
</tr>
<tr>
<td>March 24</td>
<td>First successfully generated code</td>
</tr>
<tr>
<td>April 25</td>
<td>Photon compiler finished</td>
</tr>
<tr>
<td>April 26</td>
<td>Final report finished</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akira Higaki</td>
<td>Manager, C library integration</td>
</tr>
<tr>
<td>Calum McCartan</td>
<td>Semantic Checking, Special Types</td>
</tr>
<tr>
<td>Franky Campuzano</td>
<td>Built-in functions, Presentation Slides</td>
</tr>
<tr>
<td>Phu Pham</td>
<td>Testing</td>
</tr>
</tbody>
</table>
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 -0000 - 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 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
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 add_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 23:32:08 2021 -0400 - Akira : Merge branch 'main' of https://github.com/CalumMcCartan/Photon into main
Sat Apr 24 23:29:29 2021 -0400 - Akira : Added image create
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 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 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 22:26:45 2021 -0400 - CalumMcCartan: Convert int to pint during pint assignment
Thu Apr 22 16:52:27 2021 -0400 - Calum McCartan: 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
into pints
Wed Apr 21 09:19:12 2021 -0400 - CalumMcCartan: Pint convert test
Wed Apr 21 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 21 11:23:38 2021 -0400 - CalumMcCartan: Add pint, allow to take int value
Tue Apr 20 14:21:34 2021 -0500 - Franky: Added array test for length and for loops.
Mon Apr 19 23:20:15 2021 -0500 - Franky: Added arrays. Works for floats, ints, and arrays. 4 tests added.
Thu Apr 15 18:00:11 2021 -0400 - Phu Pham: small fixes
Thu Apr 15 17:41:44 2021 -0400 - Phu Pham: min max built in added
Mon Apr 12 14:50:33 2021 -0400 - Akira: Added makefile for C image testing
Mon Apr 12 14:38:04 2021 -0400 - Akira: Added stb_image library for C and example way to use it
(no changes to Photon)
Wed Apr 7 11:23:38 2021 -0400 - CalumMcCartan: Add pint, allow to take int value
Tue Apr 6 00:27:53 2021 -0400 - Akira: Added 'func' for function declaration and changed all of the tests
to work
Mon Apr 5 12:21:52 2021 -0400 - Phu Pham: line comment added
Mon Apr 5 00:33:08 2021 -0500 - Franky: Added strings and printing strings.
Tue Mar 30 17:28:05 2021 -0400 - Calum McCartan: Merge pull request #2 from CalumMcCcartan/calum/hello-world
Tue Mar 30 17:27:39 2021 -0400 - CalumMcCcartan: Re-restart from microc
Wed Mar 24 18:45:42 2021 -0400 - Akira: Added variable assignment in parser
Wed Mar 24 16:38:03 2021 -0400 - CalumMcCcartan: Add int expression
Wed Mar 24 15:14:58 2021 -0400 - CalumMcCcartan: Fix global variable dec
Wed Mar 24 14:09:41 2021 -0400 - CalumMcCcartan: Can declare (empty) functions
Wed Mar 24 11:56:50 2021 -0400 - CalumMcCcartan: compiles with photon's scanner
Tue Mar 23 19:17:11 2021 -0400 - CalumMcCcartan: Use photons parser, scanner, ast
Tue Mar 23 19:14:13 2021 -0400 - Calum McCcartan: Merge pull request #1 from CalumMcCartan/start-from-microc
Tue Mar 23 18:31:03 2021 -0400 - CalumMcCcartan: Use microc as a starting point
Tue Mar 23 00:55:13 2021 -0500 - Franky: Merge branch 'main' of https://github.com/CalumMcCcartan/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/CalumMcCcartan/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 22:41:01 2021 -0500 - Franky: Added run line to test a sample hello world file.
Mon Mar 22 13:15:01 2021 -0400 - Akira: Added sast.ml
Fri Mar 19 09:33:54 2021 -0400 - CalumMcCcartan: fix non exhaustive matching warning
Thu Mar 18 15:56:27 2021 -0400 - Akira: Fixed function parsing and punctuation bugs
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.

<table>
<thead>
<tr>
<th>File</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanner.mll</td>
<td>Akira, Calum, Franky, Phu</td>
</tr>
<tr>
<td>photonparse.mly</td>
<td>Akira, Calum, Fanky, Phu</td>
</tr>
<tr>
<td>ast.ml</td>
<td>Calum, Franky</td>
</tr>
<tr>
<td>semant.ml</td>
<td>Akira, Calum, Franky, Phu</td>
</tr>
<tr>
<td>sast.ml</td>
<td>Akira, Franky, Calum</td>
</tr>
<tr>
<td>codegen.ml</td>
<td>Akira, Calum, Franky, Phu</td>
</tr>
<tr>
<td>util.c</td>
<td>Phu</td>
</tr>
<tr>
<td>Image.c</td>
<td>Akira, Calum, Franky, Phu</td>
</tr>
<tr>
<td>Image.h</td>
<td>Akira, Calum</td>
</tr>
</tbody>
</table>
6 Language Evolution

6.1 Photon: At the Beginning

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

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

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

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

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

pint code is very not cool sounding.

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

6.2 Rethinking Our Design Choices

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

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

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

6.3 Photon: Present-day

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

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

7.1 Source and Target Language Programs

Array Element Retrieval and Length

Photon’s source code:

```c
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:

```c
; 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.arr = getelementptr inbounds { i32*, i32* }, { i32*, i32* }* %a, i32 0, i32 1
    %p = alloca i32, i32 1028
    store i32 * %p, i32** %list.arr
    %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.arr3 = getelementptr inbounds { i32*, i32* }, { i32*, i32* }* %new.array.ptr, i32 0, i32 1
```
%p4 = alloca i32, i32 1028
store i32* %p4, i32** %list.array3
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
}

load Images
Photon’s source code:

func int main()
{
    Image img;
    int wid;
    int ht;
    img = load("Shapes.png");
wid = width(img);
ht = height(img);
print(wid);
print(ht);
save(img, "ShapesSaved.png");
return 0;
}

Generated LLVM:

; ModuleID = 'Photon'
source_filename = "Photon"

%PImage = type opaque

@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"
@str = private unnamed_addr constant [11 x i8] c"Shapes.png\00"
@str.3 = private unnamed_addr constant [16 x i8] c"ShapesSaved.png\00"

define i32 @main () {
    entry:
    % img = alloca %PImage*
    % wid = alloca i32
    %ht = alloca i32
    % load = call %PImage* @Image_load (i8* getelementptr inbounds ([11 x i8], [11 x i8]* @str, i32 0, i32 0))
    store %PImage* % load, %PImage** % img
    % img1 = load %PImage*, %PImage** % img
    % width = call i32 @Image_width(%PImage* % img1)
    store i32 % width, i32* % wid
    % img2 = load %PImage*, %PImage** % img
    % height = call i32 @Image_height(%PImage* % img2)
    store i32 % height, i32* % ht
    % wid3 = load i32, i32* % wid
    % printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 % wid3)
    % ht4 = load i32, i32* % ht
    % printf5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 % ht4)
    % img6 = load %PImage*, %PImage** % img
7.2 Test Cases

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

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

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

7.3 Test Suites & Automation

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

./testall.sh
test-add1...OK
test-arith1...OK
test-arith2...OK
test-arith3...OK
test-array1...OK
test-array2...OK
test-array3...OK
test-array4...OK
test-array5...OK
test-colour-alias...OK
test-decl-order...OK
test-fib...OK
test-float1...OK
test-float2...OK
test-float3...OK
test-for1...OK
test-for2...OK
test-func1...OK
test-func2...OK
test-func3...OK
test-func4...OK
test-func5...OK
test-func6...OK
test-func7...OK
test-func8...OK
test-func9...OK
test-gcd...OK
test-gcd2...OK
test-getpixel1...OK
test-global1...OK
test-global2...OK
test-global3...OK
test-hello...OK
test-if1...OK
test-if2...OK
test-if3...OK
test-if4...OK
test-if5...OK
test-if6...OK
test-imageadd1...OK
test-imageattr...OK
test-imagecreate...OK
test-imagedestroy...OK
test-imageflip...OK
test-imagegray...OK
test-imageinvert...OK
test-imageload1...OK
test-imagepaste1...OK
test-imagepaste2...OK
test-imagesubtract1...OK
test-int-to-float...OK
test-local1...OK
test-local2...OK
test-min-max1...OK
test-min-max2...OK
test-mixed-numeric-types...OK
test-numeric-casting...OK
test-ops1...OK
test-ops2...OK
test-pint-clamp...OK
test-pint...OK
test-pixel...OK
test-printhello...OK
test-setpixel1...OK
test-sqrt1...OK
test-sqrt2...OK
test-var1...OK
test-var2...OK
test-while1...OK
test-while2...OK
fail-array1...OK
fail-array2...OK
fail-assign1...OK
fail-assign2...OK
fail-assign3...OK
fail-assign4...OK
fail-colour-alias1...OK
fail-dead1...OK
fail-dead2...OK
fail-expr1...OK
fail-expr2...OK
fail-float1...OK
fail-float2...OK
fail-for1...OK
fail-for2...OK
fail-for3...OK
fail-for4...OK
fail-for5...OK
fail-func1...OK
fail-func2...OK
fail-func3...OK
fail-func4...OK
fail-func5...OK
fail-func6...OK
fail-func7...OK
fail-func8...OK
fail-func9...OK
fail-getpixel1...OK
fail-global1...OK
fail-global2...OK
fail-if1...OK
fail-if2...OK
fail-if3...OK
fail-imageadd...OK
fail-imagecreate1...OK
fail-imagedestroy...OK
fail-imageload1...OK
fail-imagepaste1...OK
fail-min-max1...OK
fail-missingattr...OK
fail-nomain...OK
fail-pint1...OK
fail-print...OK
fail-printb...OK
fail-printhello1...OK
fail-return1...OK
fail-return2...OK
fail-setpixel1...OK
fail-while1...OK
fail-while2...OK
make[1]: Leaving directory '/home/Photon/code'

7.4 Test Roles

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

8 Lessons Learned

8.1 Akira

I have never used a functional language like OCaml before, so learning a new subject while also learning a new programming language was a challenge for me. Specifically, one of the biggest challenges for me was to understand how
each file worked together in creating a programming language. However, slowly, I started to be able to connect the pieces that make a compiler come together. Another unique aspect about this project was the group dynamic. Since before this my only group project was in Art of Engineering, so to be honest I was a little worried about how well it would go. I have learned that good communication, clear deadlines, and small goals are the factors that encourage proper collaboration. Upon reflection, I am glad that I was able to take this class, not only because I learned a lot, but also because I can have this project as an achievement of progress.

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

8.2 Calum

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

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

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

8.3 Franky

After programming for a few years, I think it’s fair to say you understand what a programming language is, the syntax, what you can and can’t do, etc., but there’s little to inform you of the underneath mechanisms that transform the bits and pieces of code you write into digestible machine code. Taking this class really opens your eyes, especially for someone like me who knew little of what a compiler did other than yell at you for missing a semicolon.
Going into this course, I was originally obtaining a major in political science, with a concentration in computer science. However, now that we’re nearing the end, and after reading Prof. Edwards’ Actual Wisdom piece he wrote for Bwog, I’ve given serious consideration to swapping the two. Although this was one of the more complicated and technical projects I’ve ever worked on, there’s no describing the joy and relief you get from successfully compiling and running your code.

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

8.4 Phu

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

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

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

9 Future Improvements

9.1 Extra Features

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

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

9.2 Runtime Error Handling

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

10 Acknowledgements and References

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

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

11 Appendix

11.1 Code Listing

11.1.1 photon.ml

(* 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)
| 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

(* Ocamlex 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 }

37
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 | Fliteral 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 -> ">="
let string_of_uop = function
    Neg -> "-"
    Not -> "!

let rec string_of_expr = function
    PLiteral (l) = string_of_int l
    Literal (l) -> string_of_expr (Literal l)
    Alias(l) = string_of_expr (Alias l)
    Fliteral (l) = string_of_expr (Fliteral l)
    StrLiteral (l) = "" ^ l ^ ""
    BoolLit (true) -> "true"
    BoolLit (false) -> "false"
    Id(s) -> s
    Binop (e1 , o, e2) ->
        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
        string_of_expr e2
    Unop (o, e) -> string_of_uop o ^ string_of_expr e
    Assign (v, e) -> v ^ " = " ^ string_of_expr e
    Call (f, el) ->
        f ^ "(" ^ String.concat ""," (List.map string_of_expr
            el) ^ ")"
    Noexpr -> ""
    ArrayGet (id , e) -> "array_get " ^ id ^ ", " ^
        (string_of_expr e)
    ArraySize (id) -> "array_size " ^ id
    ArrayLiteral (_) -> "array_literal"
    Attr (i, a) -> i ^ "." ^ a

let rec string_of_stmt = function
    Block (stmts) ->
        "{
" ^ String.concat "" (List.map string_of_stmt
            stmts) ^ "}
    Expr (expr) -> string_of_expr expr ^ " ;" ^ 
    Return (expr) -> "return " ^ string_of_expr expr ^ " ;" ^ 
    If (e, s, Block([])) -> "if (" ^ string_of_expr e ^ 
        ")" ^ string_of_stmt s
    If (e, s1, s2) -> "if (" ^ string_of_expr e ^ 
        ")" ^ string_of_stmt s1 ^ "else" ^ string_of_stmt s2
    For (e1, e2, e3, s) ->
        "for (" ^ string_of_expr e1 ^ 
            ;" ^ string_of_expr e2 ^ 
            ;" ^ string_of_expr e3 ^ 
            ")" ^ string_of_stmt s
    While (e, s) -> "while (" ^ string_of_expr e ^ 
        ")" ^ string_of_stmt s
    ArraySet (id, e1, e2) -> "array_set " ^ id ^ 
        (string_of_expr e1) ^ " = " ^
        (string_of_expr e2)
let rec string_of_typ = function
  Int -> "int"
| Pint -> "pint"
| Bool -> "bool"
| Float -> "float"
| Void -> "void"
| String -> "string"
| Image -> "Image"
| Pixel -> "Pixel"
| Array x -> (string_of_typ x) ^ "[]"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat "", " (List.map snd
tdecl.formals) ^ "")\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)

11.1.4 photonparse.mly

/* Ocamlyacc parser for Photon */
%
open Ast
%
%token SEMI LBRACK RBRACK LPAREN RPAREN LBRACE RBRACE COMMA
PLUS MINUS TIMES DIVIDE ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT PINT BOOL FLOAT VOID
STRING FUNC
%token ARRAY_ADD ARRAY_GET ARRAY_SET ARRAY_SIZE PERIOD
LENGTH
%token IMAGE PIXEL
%token <int> LITERAL
%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
%% 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:
INT { Int } | PINT { Pint } | BOOL { Bool } | FLOAT { Float } | VOID { Void } | STRING { String } | IMAGE { Image } | PIXEL { Pixel } | typ LBRACK RBRACK { Array($1) }

typ ID SEMI { ($1, $2) }

stmt_list:
/* nothing */ { [] } | stmt_list stmt { $2 :: $1 }

stmt:
expr SEMI { Expr $1 }
| RETURN expr_opt SEMI { Return $2 }
| LBRACE stmt_list RBRACE { Block(List.rev $2) }
| IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
| IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
| FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt { For($3, $5, $7, $9) }
| WHILE LPAREN expr RPAREN stmt { While($3, $5) }
| ID LBRACK expr RBRACK ASSIGN expr SEMI { ArraySet($1, $3, $6) }
| ARRAY_ADD LPAREN ID COMMA expr RPAREN SEMI { ArrayAdd($3, $5) }

expr_opt:
/* nothing */ { Noexpr }
| expr { $1 }

expr:
LITERAL { Literal($1) }
| FLIT { Fliteral($1) }
| STRLIT { StrLiteral($1) }
| ALIAS { Alias($1) }
| BLIT { BoolLit($1) }
| ID { Id($1) }

43
11.1.5 sast.ml

(*
  Semantically-checked Abstract Syntax Tree and functions
  for printing it
  Based on MicroC

Authors:
  Akira Higaki (abh2171)
  Calum McCartan (cm4114)
  Franky Campuzano (fc2608)
*)
open Ast

type sexpr = typ * sx
and sarg = typ * sexpr
and sx =
  SLiteral of int
| SPintLit of int
| SFliteral of string
| SStrLiteral of string
| SBoolLiteral of bool
| SId of string
| SBinop of sexpr * op * sexpr
| SUnop of uop * sexpr
| SAssign of string * sexpr
| SCall of string * sarg list
| SNoexpr
| SArrayGet of typ * string * sexpr
| SArrayLiteral of typ * sexpr list
| SArraySize of typ * string

type sstmt =
  SBlock of sstmt list
| SExpr of sexpr
| SReturn of sexpr
| SIf of sexpr * sstmt * sstmt
| SFor of sexpr * sexpr * sexpr * sstmt
| SWhile of sexpr * sstmt
| SArraySet of typ * string * sexpr * sexpr
| SArrayAdd of string * sexpr

type sfunc_decl = {
  styp : typ;
  sfname : string;
  sformals : bind list;
  slocals : bind list;
  sbody : sstmt list;
}

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(\(\ " string_of_typ t \ " : " - (match e with
  SLiteral(l) -> string_of_int l
| SPintLit(l) -> string_of_int l
| SBoolLit(true) -> "true"
| SBoolLit(false) -> "false"
| SFliteral(l) -> l
| SStrLiteral(l) -> l

45
let rec string_of_sstmt = function
    SBlock stmts ->
        "\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "\n"
    | SExpr expr -> string_of_sexpr expr ^ ";
"
    | SReturn expr -> " return " ^ string_of_sexpr expr ^ ";
"
    | SIf e s sblock ->
        "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
    | SIf e s1 s2 ->
        "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
    | SFor (e1, e2, e3, s) ->
        "for (" ^ string_of_sexpr e1 ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^ string_of_sexpr e3 ^ ")\n" ^ string_of_sstmt s
    | SWhile e s ->
        "while (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
    | SArraySet (_, id, e1, e2) ->
        "array_set " ^ id ^ " \n" ^ string_of_sexpr e1 ^ ", " ^ string_of_sexpr e2
    | SArrayAdd id e ->
        "array_add " ^ id ^ ", " ^ string_of_sexpr e

let string_of_sfdecl fdecl =
    string_of_typ fdecl.styp ^ " " ^
    fdecl.sfname ^ "(" ^ String.concat "" (List.map snd fdecl.sformals) ^ ")\n" ^
    String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
    String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
    "\n"

let string_of_sprogram (vars, funcs) =
### 11.1.6 semant.ml

(*
Semantic checking for the Photon compiler
Based on MicroC

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

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
throws an exception if something is wrong.
Check each global variable, then check each function *)

let check (globals, functions) =

(* Verify a list of bindings has no void types or duplicate names *)
let check_binds (kind : string) (binds : bind list) =
List.iter (function
| (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
| _ -> ()
) binds;

let rec dups = function
| [] -> ()
| ((_,n1) :: (_,n2) :: _) when n1 = n2 -> raise
(Failure ("duplicate " ^ kind ^ " " ^ n1))
| _ :: t -> dups t
in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)

(* *** Check global variables ****)
check_binds "global" globals;

(**** Check functions ****)

(* Collect function declarations for built-in functions: no bodies *)
let built_in_decls =
  let add_bind map (name, formals', rtype) =
    StringMap.add name {
      (* object between brackets is func_decl object? *)
      typ = rtype;
      fname = name;
      formals = formals';
      locals = [];
      body = []; (* empty list *)
    } map
  in List.fold_left add_bind StringMap.empty [
    ("print", [(Int, "x")], Void);
    ("printb", [(Bool, "x")], Void);
    ("printf", [(Float, "x")], Void);
    ("prints", [(String, "x")], Void);
    ("min", [(Int, "x");(Int, "y")], Int);
    ("max", [(Int, "x");(Int, "y")], Int);
    ("sqrt", [(Float, "x")], Float);
    ("load", [(String, "x")], Image);
    ("save", [(Image, "img");(String, "fname")], Void);
    ("create", [(Int, "w");(Int, "h");(Pixel, "col")], Image);
    ("destroy", [(Image, "img")], Void);
    ("flip", [(Image, "img")], Image);
    ("to_gray", [(Image, "img")], Image);
    ("image_paste", [(Image, "target");(Image, "orig")],
      (Int, "x");(Int, "y")], Image);
    ("image_invert", [(Image, "orig")], Image);
    ("image_add", [(Image, "img1");(Image, "img2")],
      Image);
    ("image_subtract", [(Image, "img1");(Image, "img2")],
      Image);
    ("get_pixel", [(Image, "img");(Int, "x");(Int, "y")], Pixel);
    ("set_pixel", [(Image, "img");(Int, "x");(Int, "y")],
      (Pixel, "p")], Int);
    ("width", [(Image, "img")], Int);
    ("height", [(Image, "img")], Int);
    ("pixel", [(Pint, "r");(Pint, "g");(Pint, "b")],
      (Pint, "a")], Pixel);
    ("pixel_attr", [(Pixel, "p");(Int, "attr")], Pint)
  ]
(* Add function name to symbol table *)
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = "duplicate function " ^ fd.fname
  and make_err er = raise (Failure er)
  and n = fd.fname (* Name of the function *)
  in match fd with (* No duplicate functions or redefinitions of built-ins *)
    | _ when StringMap.mem n built_in_decls -> make_err built_in_err
    | _ when StringMap.mem n map -> make_err dup_err
    | _ -> StringMap.add n fd map
in

(* Collect all function names into one symbol table *)
let function_decls = List.fold_left add_func built_in_decls functions
in

(* Return a function from our symbol table *)
let find_func s =
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

let _ = find_func "main" in (* Ensure "main" is defined *)

let check_function func =
  (* Make sure no formals or locals are void or duplicates *)
  check_binds "formal" func.formals;
  check_binds "local" func.locals;

  (* Raise an exception if the given rvalue type cannot be assigned to
   the given lvalue type *)
  let check_assign lvaluet rvaluet err =
    if (lvaluet = rvaluet) then lvaluet else
      match lvaluet, rvaluet with
        | Pint, Int
        | Int, Pint
        | Float, Pint
        | Float, Int -> lvaluet
        | _ -> raise (Failure err)
in
(* Build local symbol table of variables for this function *)
let symbols = List.fold_left (fun m (ty, name) ->
  StringMap.add name ty m)
StringMap.empty (globals @ func.formals @ func.locals)

(* Return a variable from our local symbol table *)
let type_of_identifier s =
try StringMap.find s symbols
with Not_found -> raise (Failure ("undeclared identifier " ^ s))

let check_array_type id =
match (type_of_identifier id) with
  Array t -> t
| t -> raise (Failure ("check array type error,
  typ: " ^ string_of_typ t))

let combine_numeric_types t1 t2 =
if t1 = t2 then t1 else match t1, t2 with
  (* Use the more precise type *)
  | Int, Float | Float, Int -> Float
  | Pint, Float | Float, Pint -> Float
  | Pint, Int | Int, Pint -> Int
  | _ -> raise (Failure ("cannot combine numeric
types " ^ string_of_typ t1 ^ " & " ^ string_of_typ t2))

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
  | Literal l -> (Int, SLiteral l)
  | PLiteral l -> (Pint, SPintLit l)
  | Fliteral l -> (Float, SFliteral l)
  | StrLiteral l -> (String, SStrLiteral l)
  | Alias n ->
    let (r, g, b, a) = (match n with
      | "_black" -> (0, 0, 0, 255)
      | "_white" -> (255, 255, 255, 255)
      | "_grey" -> (128, 128, 128, 255)
      | "_red" -> (255, 0, 0, 255)
      | "_green" -> (0, 255, 0, 255)
      | "_blue" -> (0, 0, 255, 255)
      | "_cyan" -> (0, 255, 255, 255)
      | "_magenta" -> (255, 0, 255, 255)
      | "_yellow" -> (255, 255, 0, 255)
    )
    (Int, (r, g, b, a))
| _ -> raise (Failure ("alias " ^ n ^ " does not exist"))
| in expr (Call ("pixel", [PLiteral(r); PLiteral(g); PLiteral(b); PLiteral(a)]))
| BoolLit l -> (Bool, SBoolLit l)
| Noexpr -> (Void, SNoexpr)
| Id s -> (type_of_identifier s, SId s)
| Attr(var, attr) ->
  let lt = type_of_identifier var in
  let err = var ^ " has no attribute " ^ attr in
  (match lt with
  | Pixel ->
    (match attr with
      | "r" -> expr (Call ("pixel_attr", [Id(var); Literal(0)]))
      | "g" -> expr (Call ("pixel_attr", [Id(var); Literal(1)]))
      | "b" -> expr (Call ("pixel_attr", [Id(var); Literal(2)]))
      | "a" -> expr (Call ("pixel_attr", [Id(var); Literal(3)]))
      | _ -> raise (Failure (err)))
  | Image ->
    (match attr with
      | "width" -> expr (Call ("width", [Id(var)]))
      | "height" -> expr (Call ("height", [Id(var)]))
      | _ -> raise (Failure (err)))
  | Assign (var, e) as ex ->
    let lt = type_of_identifier var
    and (rt, e') = expr e in
    let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^ string_of_typ rt ^ " in " ^ string_of_expr ex
    in (check_assign lt rt err, SAssign (var, (rt, e')))
  | Unop (op, e) as ex ->
    let (t, e') = expr e in
    let ty = match op with
      | Neg when t = Int || t = Float -> t
      | Not when t = Bool -> Bool
      | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^ " in " ^ string_of_expr ex))
    in (ty, SUnop (op, (t, e')))
  | Binop (e1, op, e2) as e ->
    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
208       (Call("image_add", [e1; e2]))
209     | (Image, Sub, Image) -> expr
210       (Call("image_subtract", [e1; e2]))
211     | _ ->
212       let same = t1 = t2 in
213       let both_numeric =
214         ((t1 = Int) || (t1 = Pint) || (t1 = Float)) &&
215         ((t2 = Int) || (t2 = Pint) || (t2 = Float))
216       in
217       (* Determine expression type based on operator and operand types *)
218       (* Math ops require any two numeric types, logical ops require two bools *)
219       let ty = match op with
220         Add | Sub | Mult | Div when both_numeric -> combine_numeric_types t1 t2
221         Less | Leq | Greater | Geq when both_numeric -> Bool
222         Equal | Neq when same || both_numeric -> Bool
223         And | Or when same && t1 = Bool -> Bool
224         _ -> raise (Failure("illegal binary operator " ^
225           string_of_typ t1 ^ " " ^
226           string_of_op op ^ " " ^
227           string_of_typ t2 ^ " in " ^
228           string_of_expr e))
229       in (ty, SBinop((t1, e1'), op, (t2, e2'))))
230     | ArrayGet (var, e) ->
231       let (t, e') = expr e in
232       let ty = match t with
233         Int -> Int
234       in let array_type = check_array_type var
235       in (array_type, SArrayGet(array_type, var, (ty, e')))
236     | ArrayLiteral vals ->
237       (Int, SArraySize(check_array_type var, var))
238     | ArrayLiteral vals ->
239       let (t', _) = expr (List.hd vals) in
240       let map_func lit = expr lit in
241       let vals' = List.map map_func vals in
242       (Array t', SArrayLiteral(t', vals'))
241 | Call(fname, args) as call ->
242     let fd = find_func fname in
243     let param_length = List.length fd.formals in
244     if List.length args != param_length then
245         raise (Failure "expecting " ^ string_of_int
246         param_length " arguments in " ^
247         string_of_expr call))
248     else let check_call (ft, _) e =
249         let (et, e') = expr e in
250         let err = "illegal argument found " ^
251         " expected " ^ string_of_typ ft ^ " in " ^
252         string_of_expr e
253         in (check_assign ft et err, (et, e'))
254     in
255     let args' = List.map2 check_call fd.formals args
256     in (fd.typ, SCall(fname, args'))
257
258     let check_bool_expr e =
259         let (t', e') = expr e
260         and err = "expected Boolean expression in " ^
261         string_of_expr e
262         in if t' != Bool then raise (Failure err) else (t', e')
263     in
264     let check_int_expr e =
265         let (t', e') = expr e
266         and err = "expected Integer expression in " ^
267         string_of_expr e
268         in if t' != Int then raise (Failure err) else (t', e')
269     in
270     let check_match_array_type_expr l e =
271         let (t', e') = expr e
272         and err = "array type and expression type do not
273         match " ^ (string_of_typ t') ^ ", " ^ (string_of_sexpr
274         e') in
275         if t' != (check_array_type l) then raise (Failure
276         err) else (t', e')
277     in
278
279     (* Return a semantically-checked statement i.e.
280        containing sexprs *)
281     let rec check_stmt = function
282         Expr e -> SExpr (expr e)
283         | ArraySet (var, e1, e2) ->
284           SArraySet(check_array_type var, var,
285           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
289  SReturn (t, e')
290  else raise (Failure err)
291
292  (* A block is correct if each statement is correct
293   and nothing follows any Return statement. Nested blocks are
294   flattened. *)
295  | Block sl ->
296  let rec check_stmt_list = function
297  [ Return _ as s] -> [check_stmt s]
298  | Return _ :: _ -> raise (Failure "nothing
299  may follow a return")
300  | Block sl :: ss -> check_stmt_list (sl @ ss)
301  (* Flatten blocks *)
302  | s :: ss -> check_stmt s ::
303  check_stmt_list ss
304  | [] -> []
305  in SBlock(check_stmt_list sl)
306
307  in (* body of check_function *)
308  { styp = func.typ;
309  sfname = func.fname;
310  sformals = func.formals;
311  slocals = func.locals;
312  sbody = match check_stmt (Block func.body) with
313  | SBlock(sl) -> sl
314  | _ -> raise (Failure ("internal error: block didn't become a block?"))
315  }
316  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) ]

  (* 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

let type_str t =
  match t with
  | A. Int -> "int"
  | A. Pint -> "pint"
  | A. Bool -> "bool"
  | A. Float -> "float"
  | A. String -> "str"
  | _ -> raise (Failure "Invalid string map key type")

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    let init = match t with
      | A. Float -> L.const_float (ltype_of_typ t) 0.0
      | _ -> L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m
  in List.fold_left global_var StringMap.empty globals

(* built-in functions *)

let func_decl name =
  let (func_t, c_name) = match name with
    | "printf" ->
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |], "printf"
    | "min" ->
      L.function_type i32_t [| i32_t; i32_t |], "get_min"
    | "max" ->
      L.function_type i32_t [| i32_t; i32_t |], "get_max"
    | "sqrt" ->
      L.function_type float_t [| float_t |], "get_sqrt"
    | "load" ->
      L.function_type image_t [| string_t |], "Image_load"
    | "save" ->
      L.function_type i32_t [| image_t; string_t |], "Image_save"
    | "create" ->
      L.function_type image_t [| i32_t; i32_t; pixel_t |], "Image_create"
    | "width" ->
      L.function_type i32_t [| image_t |], "Image_width"
    | "height" ->
      L.function_type i32_t [| image_t |], "Image_height"
    | "destroy" ->
      L.function_type i32_t [| image_t |], "Image_free"
| "flip" -> |
| "to_gray" -> |
| "image_paste" -> |
| "image_invert" -> |
| "image_add" -> |
| "image_subtract" -> |
| "get_pixel" -> |
| "set_pixel" -> |
| "pixel" -> |
| "pixel_attr" -> |

let add_terminal builder instr =
  match L.block_terminator (L.insertion_block builder) with
  | Some _ -> ()
  | None -> ignore (instr builder)
(* array functions *)

(* the following array code partially referenced from AP++
  2018 project: *


(* ltype array_get ( array a, i32_t index ) *)

let array_get : L.llvalue StringMap.t =
  let array_get_ty m typ =
    let ltype = (ltype_of_typ typ) in

    (* define the function type *)
    let def_name = (type_str typ) in
    let def = L.define_function ("array_get" ^ def_name) (L.function_type ltype [] L.pointer_type (array_t ltype); i32_t []) the_module in

    (* create array pointer *)
    let build = L.builder_at_end context (L.entry_block def) in
    let array_ptr = L.build_alloca (L.pointer_type (array_t ltype)) "array_ptr_alloc" build in
    let _ = L.build_store (L.param def 0) array_ptr build in

    (* create index pointer *)
    let index_ptr = L.build_alloca i32_t "index_alloc" build in
    let _ = L.build_store (L.param def 1) index_ptr build in

    (* more building and allocating *)
    let array_load = L.build_load array_ptr "array_load" build in
    let array_ar_ptr = L.build_struct_gep array_load [index] "array_ar_ptr" build in
    let array_ar_load = L.build_load array_ar_ptr "array_load" build in

    (* get return value *)
    let index = L.build_load index_ptr "index_load" build in
    let array_ar_elem_ptr = L.build_gep array_ar_load [index] "list_arrary_element_ptr" build in
    let ele_val = L.build_load array_ar_elem_ptr "array_ar_element_ptr" build in
let _ = L.build_ret ele_val build in

StringMap.add def_name def m in

List.fold_left array_get_ty StringMap.empty [ A.Pint; A.Bool; A.Int; A.Float; A.String ] in

(* void array_set(array a, i32_t index, ltype value) *)
let array_set : L.llvalue StringMap.t =
let array_set_ty m typ =

let ltype = (ltype_of_typ typ) in
let def_name = (type_str typ) in
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
let build = L.builder_at_end context (L.entry_block def) in

let array_ptr = L.build_alloca (L.pointer_type (array_t ltype)) "array_ptr_alloc" build in
ignore (L.build_store (L.param def 0) array_ptr build);

let array_load = L.build_load array_ptr"array_load" build in
let array_ar_ptr = L.build_struct_gep array_load 1 "array_ar_ptr" build in
let array_ar_load = L.build_load array_ar_ptr "array_ar_load" build in

let index_element_ptr = L.build_gep array_ar_load [| L.param def 1 |] "array_ar_next_ele_ptr" build in
let _ = L.build_store (L.param def 2) index_element_ptr build in
StringMap.add def_name def m in
List.fold_left array_set_ty StringMap.empty [ A.Pint; A.Bool; A.Int; A.Float; A.String ] in

(* void array_add(array, ltype value) *)
let array_add_ty m typ =
let ltype = (ltype_of_typ typ) in
let def_name = (type_str typ) in
let def = L.define_function ("array_add" ^ def_name) (L.function_type void_t [| L.pointer_type (array_t ltype); ltype |]) the_module in
let build = L.builder_at_end context (L.entry_block def) in
let array_ptr = L.build_alloca (L.pointer_type (array_t ltype)) "array_ptr_alloc" build in
ignore(L.build_store (L.param def 0) array_ptr build);
let valPtr = L.build_alloca ltype "val_alloc" build in
ignore(L.build_store (L.param def 1) valPtr build);
let array_load = L.build_load array_ptr "array_load" build in

let array_ar_ptr = L.build_struct_gep array_load 1 "array_ar_ptr" build in
let array_ar_load = L.build_load array_ar_ptr "array_ar_load" build in
let array_size_ptr_ptr = L.build_struct_gep array_load 0 "array_size_ptr_ptr" build in
let array_size_ptr = L.build_load array_size_ptr_ptr "array_size_ptr" build in
let array_size = L.build_load array_size_ptr "array_size" build in
let next_index = array_size in
let next_element_ptr = L.build_gep array_ar_load [next_index] "next_element_ptr" build in
let next_size = L.build_add array_size (L.const_int i32_t 1) "inc_size" build in
let _ = L.build_store next_size array_size_ptr build in
let _ = L.build_store (L.build_load valPtr "val" build) next_element_ptr build in
let _ = L.build_ret_void build in
StringMap.add def_name def m in
let array_add : L.llvalue StringMap.t = List.fold_left array_add_ty StringMap.empty [ A.Pint; A.Bool; A.Int; A.Float; A.String ] in

(* i32_t array_size(array a) *)
let array_size : L.llvalue StringMap.t =
let array_size_ty m typ =
  let ltype = (ltype_of_typ typ) in
  let def_name = (type_str typ) in

  let def = L.define_function ("array_size" ~ def_name) (L.function_type i32_t [ | L.pointer_type (array_t ltype) |]) the_module in
  let build = L.builder_at_end context (L.entry_block def) in

  let array_ptr = L.build_alloca (L.pointer_type (array_t ltype)) "array_ptr_alloc" build in
  ignore(L.build_store (L.param def 0) array_ptr build);
  let array_load = L.build_load array_ptr "array_load" build in


let array_size_ptr_ptr = L. build_struct_gep array_load 0 "array_size_ptr_ptr" build in
let array_size_ptr = L. build_load array_size_ptr_ptr "array_size_ptr" build in
let array_size = L. build_load array_size_ptr "array_size" build in
ignore (L. build_ret array_size build);
StringMap.add def_name def m in
List.fold_left array_size_ty StringMap.empty [ A.Pint; A.Bool; A.Int; A.Float; A.String ] in

(* building the array *)
let init_array builder array_ptr array_type =
  (* make a size pointer and set size of the array to 0 *)
  let sizePtrPtr = L. build_struct_gep array_ptr 0 "array_size_ptr" builder in
  let sizePtr = L. build_alloca i32_t "array_size" builder in
  let _ = L. build_store (L. const_int i32_t 0) sizePtr builder in
  ignore (L. build_store sizePtr sizePtrPtr builder);
  (* create the array *)
  let array_ar_ptr = L. build_struct_gep array_ptr 1 "list.array" builder in
  let p = L. build_array_alloca (ltype_of_typ array_type) (L. const_int i32_t 1028) "p" builder in
  ignore (L. build_store p array_ar_ptr builder);
in

(*Image Functions*)

(* Define each function (arguments and return type) so we can
call it even before we've created its body *)
let function_decls : (L. llvalue * sfunc_decl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types = Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals) in
    let ftype = L. function_type (ltype_of_typ fdecl.styp) formal_types in
    StringMap.add name (L. define_function name ftype the_module, fdecl) m
  in
List.fold_left function_decl StringMap.empty functions
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sfname
  function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder in

  (* Construct the function's "locals": formal arguments
   and locally declared variables. Allocate each on the stack,
   initialize their value, if appropriate, and remember their values in
   the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p =
      L.set_value_name n p;
      let local = L.build_alloca (ltype_of_typ t) builder in
      ignore (match t with
        A.Array array_type -> init_array builder local array_type
        | _ -> ()
      );
      ignore (L.build_store p local builder);
      StringMap.add n local m
      (* Allocate space for any locally declared variables and add the
      * resulting registers to our map *)
      and add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) builder in
        ignore (match t with
          A.Array array_type -> init_array builder local_var array_type
          | _ -> ()
        );
        StringMap.add n local_var m in
    List.fold_left2 add_formal StringMap.empty fdecl.sformals

  StringMap.empty fdecl.sformals
(Array.to_list (L.params the_function))
in List.fold_left add_local formals fdecl.slocals in

let clamp v min max = if v > max then max else if v < min then min else v in

(* Return the value for a variable or formal argument. Check local names first, then global names *)
let lookup n = try StringMap.find n local_vars with Not_found -> StringMap.find n global_vars in

(* Clamp given integer between 0-255 so its ready to be cast to a pint *)
let build_pint_clamp e builder = let max = (L.const_int i32_t 255) in
  let max_cond = L.build_icmp L.Icmp.Sgt e max "compareMax" builder in
  let e = L.build_select max_cond max e "selectMax" builder in
  let min = (L.const_int i32_t 0) in
  let min_cond = L.build_icmp L.Icmp.Slt e min "compareMin" builder in
  L.build_select min_cond min e "selectMin" builder in

(* Cast an evaluated expression 'e' from type 'rt' to type 'lt' *)
let cast_expr e lt rt builder = if lt = rt then e else
  let llt = ltype_of_typ lt in
  match lt , rt with
    | A.Pint, A.Int -> let e' = build_pint_clamp e builder in L.build_trunc e' llt "pintCast" builder
    | A.Int, A.Pint -> L.build_zext e llt "intCast" builder
    | A.Float, A.Pint -> L.build_uitofp e llt "floatCast" builder
    | A.Float, A.Int -> L.build_sitofp e llt "floatCast" builder
    | _ -> raise (Failure "internal error: semant should have rejected an unsupported type conversion")
in
(* Construct code for an expression; return its value *)
let rec expr builder ((t, e) : sexpr) = match e with
| SLiteral i -> L.const_int i32_t i
| SPintLit p -> L.const_int i8_t (clamp p 0 255)
| SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
| SFliteral l -> L.const_float_of_string float_t l
| SStrLiteral s -> L.build_global_stringptr s "str" builder
| SNoexpr -> L.const_int i32_t 0
| SId s -> L.build_load (lookup s) builder
| SAssign (s, (rt, e)) ->
  let e' = expr builder (rt, e) in
  let e' = cast_expr e' t rt builder in
  ignore (L.build_store e' (lookup s) builder); e'
| SBinop ((rt1, e1), op, (rt2, e2)) ->
  let is_pint_op = (rt1 = A.Pint && rt2 = A.Pint && t = A.Pint) in
  let cast_t =
    (* If both types are pint, cast to int so we don't overflow. *)
    if is_pint_op then A.Int
    (* If binop type is a bool, then cast both expressions to float for comparison *)
    else if t = A.Bool then
      if rt1 = rt2 then rt1 else A.Float
    else t
    in
    (* Evaluate both expressions and cast to same type 'cast_t' *)
    let e1' = expr builder (rt1, e1)
    and e2' = expr builder (rt2, e2) in
    let e1' = cast_expr e1' cast_t rt1 builder
    and e2' = cast_expr e2' cast_t rt2 builder in
  let result =
    if cast_t = A.Float then (match op with
      | A.Add -> L.build_fadd
      | A.Sub -> L.build_fsub
      | A.Mult -> L.build_fmul
      | A.Div -> L.build_fdiv
      | A.Equal -> L.build_fcmpeq L.Fcmp.Oeq
      | A.Neq -> L.build_fcmpeq L.Fcmp.One
      | A.Less -> L.build_fcmplt L.Fcmp.Olt
      | A.Leq -> L.build_fcmple L.Fcmp.Ole
      | A.Greater -> L.build_fcmplt L.Fcmp.Ogt
      | A.Geq -> L.build_fcmpeq L.Fcmp.Oge
      | A.And | A.Or -> raise (Failure "internal error: semant should have rejected and/or on float")
    ) e1' e2' "floatBinop" builder
  else (match op with
    | A.Add -> L.build_add
    | A.Sub -> L.build_sub

64
A. Mult -> L.build_mul
A. Div -> L.build_sdiv
A. And -> L.build_and
A. Or -> L.build_or
A. Equal -> L.build_icmp L.Icmp.Eq
A. Neq -> L.build_icmp L.Icmp.Ne
A. Less -> L.build_icmp L.Icmp.Slt
A. Leq -> L.build_icmp L.Icmp.Sle
A. Greater -> L.build_icmp L.Icmp.Sgt
A. Geq -> L.build_icmp L.Icmp.Sge

in (e1, e2, "nonFloatBinop") builder

(* If is pint op, then cast back from int to pint *)
if is_pint_op then cast_expr result A.Pint A.Int builder
else result

| SUnop (op, ((t, _) as e)) ->
(* Unop *)
let e' = expr builder e in (match op with
A. Neg when t = A.Float -> L.build_fneg
| A. Neg -> L.build_neg
| A. Not -> L.build_not) e' "tmp" builder

(* array functions *)
| SArrayGet (array_type, id, e) ->
L.build_call (StringMap.find (type_str array_type) array_get) [| (lookup id); (expr builder e) |] "array_get" builder
| SArraySize (array_type, id) ->
L.build_call ((StringMap.find (type_str array_type)) array_size) [| (lookup id) |] "array_size" builder
| SArrayLiteral (array_type, literals) ->
let ltype = (ltype_of_typ array_type) in
let new_array_ptr = L.build_alloca (array_t ltype) "new_array_ptr" builder in
let _ = init_array builder new_array_ptr array_type in
let map_func literal =
ignore(L.build_call (StringMap.find (type_str array_type) array_add) [| new_array_ptr; (expr builder literal) |] "" builder);

in let _ = List.rev (List.map map_func literals) in
L.build_load new_array_ptr "new_array" builder

| SCall (fname, f_args) ->
let cast_arg (lt, (rt, e)) =
let e' = expr builder (rt, e) in
cast_expr e' lt rt builder

in
let args = Array.of_list (List.rev (List.map (cast_arg) (List.rev f_args))) in
let (fdef, args', result) = match fname with
  (* Built in functions with modified arguments *)
  | "printfb" |
  | "print" -> (func_decl "printf"), [] |
  | "printf" -> (func_decl "printf"), [] |
  | "printf" -> (func_decl "printf"), [] |
  | "printf" -> (func_decl "printf"), [] |
  | "printf" -> (func_decl "printf"), [] |
  | "printf" |
  | "create" | "width" |
  | "height" | "destroy" | "flip" | "to_gray" |
  | "image_paste" | "image_invert" | "image_add" | "image_subtract" |
  | "get_pixel" | "set_pixel" | "pixel" | "pixel_attr" |
  | (func_decl fname), args, fname |
  | _ -> |
  let (fdef, fdecl) = StringMap.find fname function_decls in |
  let result = (match fdecl.styp with |
    | A.Void -> "" |
    | _ -> fname ^ "_result") |
  in L.build_call fdef args' result builder in

(* Build the code for the given statement; return the builder for |
  the statement's successor (i.e., the next instruction will be built |
  after the one generated by this call) *)

let rec stmt builder = function |
  | SBlock sl -> List.fold_left stmt builder sl |
  | SArrayAdd (id, e) -> ignore(L.build_call (StringMap.find (type_str (fst e)) array_add) [| (lookup id); (expr builder e) |] "" builder); builder |
  | SArraySet (array_type, id, e1, e2) -> ignore(L.build_call (StringMap.find (type_str array_type) array_set) [| (lookup id); (expr builder e1); (expr builder e2) |] "" builder); builder |
  | SExpr e -> ignore(expr builder e); builder |
  | SReturn (t, e) ->
ignore(match fdecl.styp with
  (* Special "return nothing" instr *)
  | A.Void -> L.build_ret_void builder
  (* Build return statement *)
  | _ -> let e' = expr builder (t, e) in
    let e' = cast_expr e' fdecl.styp t builder in
    L.build_ret e' builder
); builder
| SIf (predicate, then_stmt, else_stmt) ->
  let bool_val = expr builder predicate in
  let merge_bb = L.append_block context "merge"
  the_function in
  let build_br_merge = L.build_br merge_bb in (*
  partial function *)

  let then_bb = L.append_block context "then"
  the_function in
  add_terminal (stmt (L.builder_at_end context
  then_bb) then_stmt)
  build_br_merge;

  let else_bb = L.append_block context "else"
  the_function in
  add_terminal (stmt (L.builder_at_end context
  else_bb) else_stmt)
  build_br_merge;

  ignore(L.build_cond_br bool_val then_bb else_bb
  builder);
  L.builder_at_end context merge_bb
| SWhile (predicate, body) ->
  let pred_bb = L.append_block context "while"
  the_function in
  ignore(L.build_br pred_bb builder);

  let body_bb = L.append_block context "while_body"
  the_function in
  add_terminal (stmt (L.builder_at_end context body_bb)
  body)
  (L.build_br pred_bb);

  let pred_builder = L.builder_at_end context pred_bb in
  let bool_val = expr pred_builder predicate in

  let merge_bb = L.append_block context "merge"
  the_function in
  ignore(L.build_cond_br bool_val body_bb merge_bb
  pred_builder);
  L.builder_at_end context merge_bb
(* Implement for loops as while loops *)
| SFor (e1, e2, e3, body) ->
  stmt builder ( SBlock [SExpr e1 ; SWhile (e2, SBlock
  [body ; SExpr e3]) ] )
in
(* Build the code for each statement in the function *)
let builder = stmt builder (SBlock fdecl.sbody) in
(* Add a return if the last block falls off the end *)
add_terminal builder (match fdecl.styp with
  A.Void -> L.build_ret_void
  | A.Float -> L.build_ret (L.const_float float_t 0.0)
  | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in
List.iter build_function_body functions;
the_module

11.1.8 Image.c

/*
 * Image library for Photon that implements the stb_image library.
 * Image_load, Image_free and Image_save are written by the stb_image
 * library team, the rest is written by the Photon team.
 */

#include "Image.h"
#include "utils.h"
#include <math.h>

#define STB_IMAGE_IMPLEMENTATION
#define "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#define "stb_image/stb_image_write.h"

Image* Image_load(const char *fname) {
  Image* img = malloc(sizeof(Image));
if((img->data = stbi_load(fname, &img->width, 
    &img->height, &img->channels, 4)) != NULL) {
    img->size = img->width * img->height * 
    img->channels;
    img->allocation_ = STB_ALLOCATED;
} else {
    printf("Failed to load image %s\n", fname);
}
return img;
}

void Image_save(Image * img, const char * fname) {
    if( str_ends_in (fname, ".jpg") || str_ends_in (fname, ".JPG") || 
        str_ends_in (fname, ".jpeg") || 
        str_ends_in (fname, ".JPEG")) {
        stbi_write_jpg (fname, img->width, img->height, 
                        img->channels, img->data, 100);
    } else if( str_ends_in (fname, ".png") || 
               str_ends_in (fname, ".PNG")) {
        stbi_write_png (fname, img->width, img->height, 
                        img->channels, img->data, img->width * img->channels);
    } else {
        ON_ERROR_EXIT(false, "");
    }
}

int Image_width(Image * img) {
    return img->width;
}

int Image_height(Image * img) {
    return img->height;
}

int get_position(int width, int channels, int x, int y) {
    return (x + (width * y)) * (channels);
}

Pixel get_pixel(const Image * img, int x, int y) {
    int pos = get_position(img->width, img->channels, x, y);
    unsigned char *p = img->data;
    unsigned char pixchar = p[pos];
    return pixel(p[pos], p[pos + 1], p[pos + 2], p[pos + 3]);
}

int set_pixel(Image * img, int x, int y, Pixel pix) {
    unsigned char *p = img->data;
    int pos = get_position(img->width, img->channels, x, y);
Pixel pixel(uint8_t red, uint8_t green, uint8_t blue, 
    uint8_t alpha) {
    Pixel p;
    p.r = red;
    p.g = green;
    p.b = blue;
    p.a = alpha;
    return p;
}

uint8_t pixel_attr(Pixel p, int attr) {
    switch(attr) {
    case 0: return p.r;
    case 1: return p.g;
    case 2: return p.b;
    case 3: return p.a;
    default: printf("Internal error: pixel has no \%d\n attr", attr);
    }
    return 0;
}

Image* Image_create(int width, int height, Pixel col) {
    Image* img = malloc(sizeof(Image));
    size_t size = width * height * 4;
    img->data = malloc(size);

    if(img->data != NULL) {
        img->width = width;
        img->height = height;
        img->size = size;
        img->channels = 4;
        img->allocation_ = SELF_ALLOCATED;
    }

    for(unsigned char *p = img->data; p != img->data + 
        img->size; p += img->channels) {
        *p = col.r;
        *(p + 1) = col.g;
        *(p + 2) = col.b;
        *(p + 3) = col.a;
113        } return img;
114    }
115
116    void Image_free(Image *img) {
117        if(img->allocation_ != NO_ALLOCATION && img->data != NULL) {
118            if(img->allocation_ == STB_ALLOCATED) {
119                stbi_image_free(img->data);
120            } else {
121                free(img->data);
122            }
123            img->data = NULL;
124            img->width = 0;
125            img->height = 0;
126            img->size = 0;
127            img->allocation_ = NO_ALLOCATION;
128        }
129    }
130
131    Image* Image_paste( Image *gray, const Image *orig, int x, int y ) {
132        //ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
133            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 +=
140            gray->channels) {
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            *pg = *p;
151            *(pg + 1) = *(p+1);
152            *(pg + 2) = *(p+2);
153            if(orig->channels == 4) {
154        }
*\( pg + 3) = *(p + 3); *
\}
i+=orig->channels;
} return gray;
}

Image* Image_invert(Image *orig) {
//ON_ERROR_EXIT(!(orig->allocation != NO_ALLOCATION &&
orig->channels >= 3), "The input image must have at
least 3 channels.");

uint8_t pix;
int8_t maxval = 255;
int i = 0;
for (unsigned char *p = orig->data; p != orig->data +
orig->size; p += orig->channels) {
    pix = *p;
    *p = maxval - pix;
    pix = *(p + 1);
    *(p + 1) = maxval - pix;
    pix = *(p + 2);
    *(p + 2) = maxval - pix;
}
return orig;
}

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

if (w <= img2->width && h <= img2->height) {
    for (int c = 0; c < img2->channels; c++) {
        gray->data[g_idx + c] -= img2->data[img2_idx + c];
    }
    img2_idx += img2->channels;
}

if (w == gray->width) {
    w = 1;
    h += 1;
} else {
    w += 1;
}

gray = Image_create(img2->width, img2->height, pixel(0, 0, 0, 255));
for (int g_idx = 0, img1_idx = 0, img2_idx = 0;
    g_idx < gray->size; g_idx += gray->channels, img2_idx +=
    img2->channels) {
    // printf("%d", g_idx);
    for (int c = 0; c < img2->channels; c++) {
        gray->data[g_idx + c] = img2->data[img2_idx + c];
    }
    if (w <= img1->width && h <= img1->height) {
        for (int c = 0; c < img1->channels; c++) {
            gray->data[g_idx + c] -=
            img1->data[img1_idx + c];
        }
        img1_idx += img1->channels;
    }
    if (w == gray->width) {
        w = 1;
        h += 1;
    } else {
        w += 1;
    }
}

return gray;
// ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
orig->channels >= 3), "The input image must have at
least 3 channels.");

Image* gray = Image_create(orig->width, orig->height,
pixel(0, 0, 0, 255));
ON_ERROR_EXIT(gray->data == NULL, "Error in creating
the image");

uint8_t gray_p;

for(unsigned char * p = orig->data, *pg = gray->data ; p
!= orig->data + orig->size; p += orig->channels, pg +=
gray->channels) {
    gray_p = (uint8_t)((*p + *(p + 1) + *(p + 2))/3.0);
    *pg = gray_p;
    *(pg + 1) = gray_p;
    *(pg + 2) = gray_p;
    if(orig->channels == 4) {
        *(pg + 3) = *(p + 3);
    }
}
return gray;

Image* Image_flip(const Image *orig) {
    // ON_ERROR_EXIT(!(orig->allocation_ != NO_ALLOCATION &&
orig->channels >= 3), "The input image must have at
least 3 channels.");

    int channels = 4;
    Image* flipped = Image_create(orig->width, orig->height, pixel(0, 0, 0, 255));
    ON_ERROR_EXIT(flipped->data == NULL, "Error in creating
the image");
    int index = 0;
    int flippedIndex = 0;

    for(int y = 0; y < orig->height; y++) {
        for (int x = 0; x < orig->width; x++){
            index = (x + (orig->width * y)) * channels;
            flippedIndex = (((orig->width - x) +
(orig->width * y)) * channels;
            for (int c = 0; c < channels; c++) {
                flipped->data[flippedIndex + c] =
orig->data[index + c];
            }
        }
    }
 return flipped;
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 {


Pixel pix;

Image* Image_load(const char *fname);
void Image_save(Image *img, const char *fname);
Image* Image_create(int width, int height, Pixel col);
void Image_free(Image *img);
Image* Image_to_gray(const Image *orig);
Image* Image_flip(const Image *orig);
Image* Image_add(Image *img1, Image *img2);
Image* Image_subtract(Image *img1, Image *img2);
Pixel pixel(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha);

11.1.10  utils.c

/*
Utility functions for Photon.
Based on MicroC

Authors:
Phu D Pham (pdp2121)
*/

#include <stdio.h>
#include <math.h>

int get_max(int x, int y) {
    if (x >= y) {
        return x;
    } else {
        return y;
    }
}

int get_min(int x, int y) {
    if (x <= y) {
        return x;
    } else {
        return y;
    }
}

double get_sqrt(double x) {
    return sqrt(x);
}

#ifdef BUILD_TEST
int main()
#endif
```c
{ 
    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
```

78
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 -- \n  ocamlbuild -use-ocamlfind photon.native

# "make clean" removes all generated files
.PHONY : clean
clean :
  rm -rf testall.log ocamllvm *.diff _build photon.native
  utils.o Image.o images-out
  ocamlbuild -clean
  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.ml 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
# Regression testing script for Photon
# Step through a list of files
# Compile, run, and check the output of each expected-to-work test
# Compile and check the error of each expected-to-fail test
# Copy images used in tests into root dir
cd images
{
    for f in *.png
do
cp -v "${f}" .."${f}"
done
} > /dev/null 2>&1
cd ..

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the photon compiler. Usually "./photon.native"
# Try ".build/photon.native" if ocamlbuild was unable to create a symbolic link.
Photon="./photon.native"
#Photon=".build/photon.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
keep=0

Usage() {
    echo "Usage: testall.sh [options] [.phn files]"
    echo "-k Keep intermediate files"
    echo "-h Print this help"
    exit 1
}

80
SignalError() {
    if [ $error -eq 0 ]; then
        echo "FAILED"
        error=1
    fi
    echo "$1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 >"$3" 1>&2
    diff -b "$1" "$2" >"$3" 2>&1 || {
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*
        return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\///'
    s/.phn//`
    reffile=`echo $1 | sed 's/.phn$//'`
    basedir=`echo $1 | sed 's/\/[~\/]*/$/'`
    echo -n "$basename..."
echo 1>&2
echo "##### Testing $basename" 1>&2

generatedfiles=""
generatedfiles="$generatedfiles ${basename}.ll
${basename}.s ${basename}.exe ${basename}.out" &&
Run "$Photon" "$1" ">" "$({basename}.ll" "&"
Run "$LLC" "-relocation-model=pic" "$({basename}.ll" ">"
"${{basename}}.s" &&
Run "$CC" "-o" "${{basename}}.exe" "$({basename}).s"
"utils.o Image.o -lm" &&
Run "/$({basename}).exe" > "$({basename}).out" &&
Compare ${{basename}}.out ${reffile}.out ${basename}.diff

# Report the status and clean up the generated files

if [ $error -eq 0 ] ; then
if [ $keep -eq 0 ] ; then
  rm -f $generatedfiles
fi
  echo "OK"
else
  echo "##### FAILED" 1>&2
globalerror=$error
  fi
}

CheckFail() {
  error=0
  basename=`echo $1 | sed 's/.*/\///
  s/.phn//'
  reffile=`echo $1 | sed 's/.phn$//'`
basedir="`echo $1 | sed 's/\/[\/]*/$// .'`/
  echo -n "$basename..."
  echo 1>&2
  echo "##### Testing $basename" 1>&2
  generatedfiles=""
  generatedfiles="$generatedfiles ${basename}.err
  ${basename}.diff" &&
  RunFail "$Photon" "<" $1 "2>>" "$({basename}).err" "&>
  "$globallog &&
  Compare ${{basename}}.err ${reffile}.err ${{basename}}.diff
  # Report the status and clean up the generated files
if [ $error -eq 0 ]; then
  if [ $keep -eq 0 ]; then
    rm -f $generatedfiles
  fi
  echo "OK"
  echo "####### SUCCESS" 1>&2
else
  echo "####### FAILED" 1>&2
  globalerror=$error
  fi
}

while getopts kdpsh c; do
  case $c in
    k) # Keep intermediate files
      keep=1
      ;;
    h) # Help
      Usage
      ;;
    esac
  done
  shift `expr $OPTIND - 1`

  LLIFail() {
    echo "Could not find the LLVM interpreter "$LLI"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
  }

  which "$LLI" >> $globallog || LLIFail
  if [ ! -f utils.o ]
  then
    echo "Could not find utils.o"
    echo "Try \"make utils.o\""
    exit 1
  fi
  if [ ! -f Image.o ]
  then
    echo "Could not find Image.o"
    echo "Try \"make Image.o\""
    exit 1
  fi
  if [ $# -ge 1 ]
then
  files=$@
else
  files="tests/test-*.phn tests/fail-*.phn"
fi
for file in $files
do
case $file in
  *test-*)
    Check $file 2>> $globallog
    ;;
  *fail-*)
    CheckFail $file 2>> $globallog
    ;;
  *)
    echo "unknown file type $file"
    globalerror=1
    ;;
esac
done

# Move output images to images-out
rm -f -R images-out
mkdir -p images-out
for f in *.png
do
  if ! test -f images/"$f"; then
    mv "$f" images-out/"$f"
  fi
  done
rm *.png
exit $globalerror
11.1.14 Sample Images

Shapes.png:

greyTest.png:

edwards.png:
11.2 Test Scripts

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

```plaintext
fail-array1.phn

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

func int main()
{
    int i;
    bool b;
    i = 42;
    i = 10;
    b = true;
    b = false;
    i = false; # Fail: assigning a bool to an integer
}

func int main()
{
    int i;
    bool b;
    b = 48; # Fail: assigning an integer to a bool
}

func void myvoid()
{
    return;
}
func int main()
{
    int i;
    i = myvoid(); # Fail: assigning a void to an integer
}

fail-assign4.phn

func int main()
{
    pint i;
    i = 0.5; # Fail: cant assign float to pint
}

fail-colour-alias1.phn

func int main()
{
    Pixel a;
    a = _silver;
    print(a.r);
    print(a.g);
    print(a.b);
    print(a.a);
    return 0;
}

fail-dead1.phn

func int main()
{
    int i;
    i = 15;
    return i;
    i = 32; # Error: code after a return
}

fail-dead2.phn
func int main()
{
    int i;
    
    int 15;
    return i;
}
int 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()
{

return 0;
}

fail-float1.phn
func int main()
{
  -3.5 && 1; # Float with AND?
    return 0;
}

fail-float2.phn
func int main()
{
  -3.5 && 2.5; # Float with AND?
    return 0;
}

fail-for1.phn
func int main()
{
  int i;
    for (; true ;) {} # OK: Forever
    for (i = 0 ; i < 10 ; i = i + 1) {
        if (i == 3) return 42;
    }
    for (j = 0; i < 10 ; i = i + 1) {} # j undefined
    return 0;
}

fail-for2.phn
func int main()
{
  int i;
    for (i = 0; j < 10 ; i = i + 1) {} # j undefined
    return 0;
fail-for3.phn

```go
func int main()
{
    int i;
    for (i = 0; i ; i = i + 1) {} # i is an integer, not Boolean
    return 0;
}
```

fail-for4.phn

```go
func int main()
{
    int i;
    for (i = 0; i < 10 ; i = j + 1) {} # j undefined
    return 0;
}
```

fail-for5.phn

```go
func int main()
{
    int i;
    for (i = 0; i < 10 ; i = i + 1) {
        foo(); # Error: no function foo
    }
    return 0;
}
```

fail-func1.phn

```go
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

```plaintext
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

```plaintext
func void foo(int a, bool b) {
}
func int main()
{
    foo(42, true);
    foo(42); # Wrong number of arguments
}
```

fail-func7.phn

```plaintext
func void foo(int a, bool b) {
}
func int main()
{
    foo(42, true);
    foo(42, true, false); # Wrong number of arguments
}
```

fail-func8.phn

```plaintext
```
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;
p = get_pixel(img, x, y);
print(p.r);
print(p.g);
print(p.b);
print(p.a);
return 0;

fail-global1.phn
int c;
bool b;
void a; # global variables should not be void
func int main()
{
  return 0;
}

fail-global2.phn
int b;
bool c;
int a;
int b; # Duplicate global variable
func int main()
{
  return 0;
}

fail-if1.phn
func int main()
{
  if (true) {} # Error: non-bool predicate
  if (false) {} else {}
  if (42) {} # Error: non-bool predicate
}
fail-if2.phn

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

fail-if3.phn

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

fail-imageadd.phn

```go
func int main() {
    Image img1;
    img1 = load("Shapes.png");
    img1 + 5;
    return 0;
}
```

fail-imagecreate1.phn

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

  img = load("Shapes.png");
  wid = width(img);
  ht = height(img);
  print(wid);
  print(ht);
  #img1 undeclared
  destroy(img1);

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

  return 0;
}

func int main()
{
  float img;
  int wid;
  int ht;

  #load non existed image should returns error
  img = load("Shapes.png");
}
fail-imagepaste1.phpn

```php
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.phpn

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

fail-missingattr.phpn

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

fail-return2.phn
1 func void foo()
2 {
3     if (true) return 42; # Should return void
4     else return;
5 }
6
7 func int main()
8 {
9     return 42;
10 }

fail-setpixel1.phn
1 func int main()
2 {
3     # testing set_pixel
4     Image img;
5     int index;
6     int position;
7     int channels;
8
9     int p1;
10     int x;
11     int y;
12
13     img = load("Shapes.png");
14
15     # for getting a specific pixel, input the width * height
cordinate you want
16     x = 0;
17     y = 0;
18     # set pixel to an int should return error
19     p1 = 30;
20
21     # Set pixel
22     set_pixel(img, x, y, p1);
23     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;

fail-while1.phn

func int main()
{
    int i;
    while (true) {
        i = i + 1;
    }
    while (42) { # Should be boolean
        i = i + 1;
    }
}

fail-while2.phn

func int main()
{
    int i;
    while (true) {
        i = i + 1;
    }
    while (true) {
        foo(); # foo undefined
    }
}
test-add1.phn

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

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

test-arith1.phn

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

test-arith2.phn

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

test-arith3.phn

```plaintext
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;
}

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;
}

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

    return 0;
}
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

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

test-float2.phn

```c
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

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

```c
func int main()
{
    float c;
    float d;
    ```
c = 42.0;
d = 3.14159;
testfloat(c, d);
testfloat(d, d);
return 0;
}

test-for1.phn
__func int main()
{
    int i;
    for (i = 0 ; i < 5 ; i = i + 1) {
        print(i);
    }
    print(42);
    return 0;
}

test-for2.phn
__func int main()
{
    int i;
    i = 0;
    for ( ; i < 5; ) {
        print(i);
        i = i + 1;
    }
    print(42);
    return 0;
}

test-func1.phn
__func int add(int a, int b)
{
    return a + b;
}

__func int main()
{
    int a;


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

---

test-func2.phn

\# Bug noticed by Pin-Chin Huang

3 \texttt{func int fun(int x, int y) }
4 \{ 
5 \texttt{return 0;}
6 \}
7
8 \texttt{func int main()}
9 \{ 
10 \texttt{int i;}
11 \texttt{i = 1;}
12 \texttt{fun(i = 2, i = i+1);}
13 \texttt{print(i);}
14 \texttt{return 0;}
15 \}

---

test-func3.phn

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

---

test-func4.phn

1 \texttt{func int add(int a, int b) }
2 \{ 
3 \texttt{int c;}
4
c = a + b;
return c;
}

func int main()
{
    int d;
    d = add(52, 10);
    print(d);
    return 0;
}

---

test-func5.phn

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

d = add(52, 10);
print(d);
return 0;

---

test-func6.phn

func int bar(int a, bool b, int c) { return a + c; }

func int main()
{
    print(bar(17, false, 25));
    return 0;
}

---

test-func7.phn

int a;

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

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));
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(14, 21));
    print(gcd(8, 36));
    print(gcd(99, 121));
    return 0;
}

func int main()
{
    # whatever
    Image img;
    int index;
    Pixel p;
    int x;
    int y;
    int position;
    int channels;
    Pixel ps;
    img = load("edwards.png");
    # for getting a specific pixel, input what you want
    x = 1;
    y = 0;
    p = get_pixel(img, x, y);
    print(p.r);
    print(p.g);
test-global1.phn

```plaintext
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

```plaintext
bool i;

func int main()
{
    int i; # Should hide the global i
```
7    i = 42;
8    print(i + i);
9    return 0;
10  }

1
test-global3.phn
2
test-hello.phn
3
test-if1.phn
4
test-if2.phn
func int main()
{
    if (false) print(42);
    print(17);
    return 0;
}

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;
}

func int cond(bool b)
{
```c
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;
}
```

```c
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

```phn
func int main() {
    Image img;
    img = load("Shapes.png");
    print(img.width);
    print(img.height);
    return 0;
}
```

test-imagecreate.phn

```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

```phn
func int main() {
    Image img;
    int wid;
    int ht;
    img = load("Shapes.png");
    wid = width(img);
    ht = height(img);
    ...
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;
}

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

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

    x = y = 0;
    newimg = image_paste(imgshapes, imgedwards, 0,0);

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

    save(newimg, "ImgPasteTest.png");

    return 0;
}
```

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

    save(img, "ShapesSaved.png");
    return 0;
}
```

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

    test-imagepaste2.phn
```

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

    test-imagepaste2.phn
```

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

    test-imagepaste2.phn
```
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;
}

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

```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

```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

```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;
    int p;
    float f;
p = 10;
f = 0.5;
```
func int main() {
    int i;
    pint p;
    float f;
    p = 3;
    i = returnPintAsInt();
    f = 9;
    print(p);
    print(i);
    printf(f);
    p = i;
    i = p;
    f = i + 1;
    print(p);
    print(i);
    printf(f);
    f = p;
    printf(f);
    useIntForFloatArg(i);
    return 0;
}

func int returnPintAsInt() {
    pint p;
    p = 6;
    return p;
}

func void useIntForFloatArg(float f) {
```
test-ops1.phn

```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

```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);
```
```
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 func int printInt(pint p) {
46     print(p);
47     return p;
48 }
test-printhello.phn

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

test-setpixel1.phn

```c
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

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

test-sqrt2.phn

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

test-var1.phn

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

test-var2.phn

```c
int a;

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

func int main()
{
    foo(73);
    print(a);
    return 0;
}
func int main() {
    int i;
    i = 5;
    while (i > 0) {
        print(i);
        i = i - 1;
    }
    print(42);
    return 0;
}

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;
}